



PATENT ABSTRACTS OF JAPAN

(11) Publication number: **2002175344 A**(43) Date of publication of application: **21.06.02**

(51) Int. Cl.

G06F 17/50
G01R 31/28
G06F 11/26

(21) Application number: **2001274491**(22) Date of filing: **11.09.01**(30) Priority: **17.10.00 US 2000 688745**(71) Applicant: **NEC CORP**(72) Inventor: **LESLIE J FRENCH**

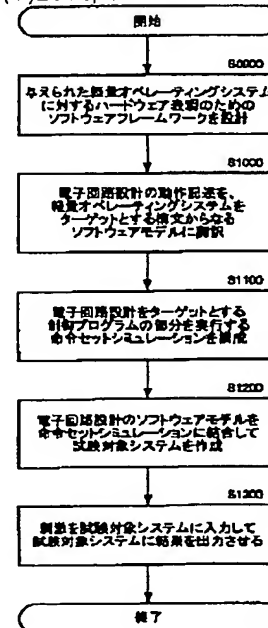
(54) **CO-VALIDATION METHOD BETWEEN
 ELECTRONIC CIRCUIT AND CONTROL
 PROGRAM**

COPYRIGHT: (C)2002,JPO

(57) Abstract:

PROBLEM TO BE SOLVED: To use a light-weight operating system on a message base as a base for co-design (cooperative design) and co-simulation (cooperative simulation) of hardware/software.

SOLUTION: Circuit design is accepted operation description and is mapped to software processes. A VHDL process is translated by using a rule set designed for a specified light weight OS to be used. Hardware processes, mapped to the software mutually communicate state changes through message transfer primitive of the light weight OS, using an interface library. Integrated co-simulation environment of the software (to be written, for example, in C or assembler language) and the hardware (to be written, for example, in VHDL and a subset of C) is obtained by the addition of an instruction set simulator which operates under the same OS is added.



(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2002-175344

(P2002-175344A)

(43) 公開日 平成14年6月21日 (2002.6.21)

(51) Int.Cl. ⁷	識別記号	F I	テマコード [*] (参考)
G 0 6 F 17/50	6 6 4	G 0 6 F 17/50	6 6 4 A 2 G 1 3 2
			6 6 4 J 5 B 0 4 6
G 0 1 R 31/28		11/26	5 B 0 4 8
G 0 6 F 11/26		G 0 1 R 31/28	F

審査請求 有 請求項の数25 O L (全 33 頁)

(21) 出願番号 特願2001-274491 (P2001-274491)

(22) 出願日 平成13年9月11日 (2001.9.11)

(31) 優先権主張番号 09/688745

(32) 優先日 平成12年10月17日 (2000.10.17)

(33) 優先権主張国 米国 (U S)

(71) 出願人 000004237

日本電気株式会社

東京都港区芝五丁目7番1号

(72) 発明者 レスリー・ジェイ・フレンチ

アメリカ合衆国, ニュージャージー

08540 プリンストン, 4 インディペン

デンス ウエイ, エヌ・イー・シー・ユ

ー・エス・エー・インク内

(74) 代理人 100097157

弁理士 桂木 雄二

F ターム (参考) 2G132 AC09

5B046 AA08 BA03 JA05

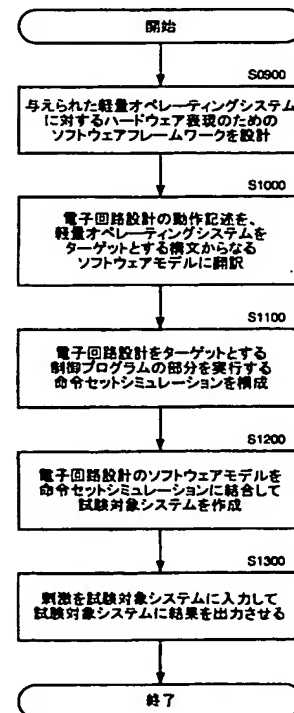
5B048 DD14

(54) 【発明の名称】 電子回路と制御プログラムとのコバリデーション方法

(57) 【要約】

【課題】 ハードウェア/ソフトウェア・コデザイン (協調設計) およびコシミュレーション (協調シミュレーション) の基礎として、メッセージベースの軽量オペレーティングシステムを利用する。

【解決手段】 回路設計を動作記述として受け入れ、ソフトウェアプロセスにマッピングする。VHDL プロセスは、使用される特定の軽量OS用に設計された規則セットを用いて翻訳される。ソフトウェアにマッピングされたハードウェアプロセスは、インタフェースライブラリを用いて軽量OSのメッセージ受渡しプリミティブを通じて状態変化を通信しあう。同じOSの下で動作する命令セットシミュレータが付加されることにより、ソフトウェア (例えばCやアセンブラで書かれる) とハードウェア (例えばVHDLやCのサブセットで書かれる) の統合コシミュレーション環境が得られる。



【特許請求の範囲】

【請求項 1】 電子回路と該電子回路をターゲットとする制御プログラムとのコバリデーションを行う方法において、前記電子回路および前記制御プログラムが軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされ、前記方法は、
前記軽量コンピュータシステムオペレーティング環境をターゲットとする構文からなるソフトウェアモデルに、前記電子回路の動作シミュレーションを翻訳するステップと、
命令セットシミュレーションが前記電子回路をターゲットとする制御プログラムの一部を実行するように、サイクル精度の命令セットシミュレーションを構成するステップと、
前記電子回路のソフトウェアモデルを前記命令セットシミュレーションに結合して、前記軽量コンピュータオペレーティングシステム環境で実行される試験対象システムを作成するステップと、
前記試験対象システムに刺激を入力することにより、前記試験対象システムに結果を出力させるステップと、
を有することを特徴とする電子回路と制御プログラムとのコバリデーション方法。

【請求項 2】 前記動作シミュレーションを翻訳するステップは、複数の規則を用いて、前記動作シミュレーションを、前記軽量コンピュータシステムオペレーティング環境をターゲットとする構文に変換するステップを有することを特徴とする請求項 1 記載の方法。

【請求項 3】 前記複数の規則のうちの 1 つは、動作シミュレーション代入を、前記軽量コンピュータシステムオペレーティング環境をターゲットとする内部変数構文に変換することを特徴とする請求項 2 記載の方法。

【請求項 4】 前記複数の規則のうちの 1 つは、動作シミュレーション信号代入を、前記軽量コンピュータシステムオペレーティング環境をターゲットとする外部変数構文に変換することを特徴とする請求項 2 記載の方法。

【請求項 5】 前記複数の規則のうちの 1 つは、動作シミュレーションウェイトポイントを、前記軽量コンピュータシステムオペレーティング環境をターゲットとするウェイト構文に変換することを特徴とする請求項 2 記載の方法。

【請求項 6】 前記複数の規則のうちの 1 つは、動作シミュレーション制御構文を、前記軽量コンピュータシステムオペレーティング環境をターゲットとする制御構文に変換することを特徴とする請求項 2 記載の方法。

【請求項 7】 前記サイクル精度の命令セットシミュレーションを構成するステップは、中央処理装置をターゲットとするシミュレーションを構成するステップを含むことを特徴とする請求項 1 記載の方法。

【請求項 8】 前記命令セットシミュレーションを構成

するステップは、グルーロジックを通じてターゲット中央処理装置に接続されたハードウェアデバイスを表すようにアドレス範囲をマッピングするステップを含むことを特徴とする請求項 7 記載の方法。

【請求項 9】 前記命令セットシミュレーションを構成するステップは、ターゲット中央処理装置のキャッシュ動作をエミュレートするステップを含むことを特徴とする請求項 7 記載の方法。

【請求項 10】 前記命令セットシミュレーションを構成するステップは、ターゲット中央処理装置の命令スケジューリングをエミュレートするステップを含むことを特徴とする請求項 7 記載の方法。

【請求項 11】 前記軽量コンピュータシステムオペレーティング環境をターゲットとする構文は、前記軽量コンピュータシステムオペレーティング環境をターゲットとするインタフェースライブラリの一部を含むことを特徴とする請求項 1 記載の方法。

【請求項 12】 電子回路と、該電子回路をターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムにおいて、
前記電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされ、
前記コンピュータシステムは、
すべてのグローバル信号の状態を管理するソフトウェア手段と、
クロック信号を生成するソフトウェア手段と、
イベントをウェイトするプロセスキューを管理するソフトウェア手段と、
所定のタイミング間隔で少なくとも 1 つのプロセスを実行するソフトウェア手段と、
中央プロセス制御を行うソフトウェア手段と、
を有することを特徴とするコンピュータシステム。

【請求項 13】 前記グローバル信号に対する状態変化を表示するディスプレイを駆動するソフトウェア手段をさらに有することを特徴とする請求項 12 記載のコンピュータシステム。

【請求項 14】 電子回路と、該電子回路をターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムにおいて、
前記電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされ、
前記コンピュータシステムは、
プロセッサと、
前記コンピュータシステムが中央制御プロセスを実行することを可能にするように適応したソフトウェア命令を含むメモリと、を有し、
前記中央制御プロセスは、
複数のグローバル信号の状態を管理する信号メンテナサ

ブプロセスと、
 ソフトウェアモデルおよび命令セットシミュレータによって使用されるクロック信号を生成するクロック信号ジェネレータサブプロセスと、
 ソフトウェアモデルおよび命令セットシミュレータからのイベントをウェイトするプロセスキューを管理するキューメンテナサブプロセスと、
 所定のタイミング間隔を生成するタイマサブプロセスと、
 少なくとも前記信号メンテナサブプロセス、クロック信号ジェネレータサブプロセス、キューメンテナサブプロセスおよびタイマサブプロセスの実行を制御するコントローラサブプロセスと、を有することを特徴とするコンピュータシステム。

【請求項15】 前記所定のタイミング間隔は、1ナノ秒以上の時間間隔をシミュレートすることを特徴とする請求項14記載のコンピュータシステム。

【請求項16】 前記所定のタイミング間隔は、1ナノ秒未満の時間間隔をシミュレートすることを特徴とする請求項14記載のコンピュータシステム。

【請求項17】 前記グローバル信号に対する状態変化を表示するグローバルディスプレイサブプロセスをさらに有することを特徴とする請求項14記載のコンピュータシステム。

【請求項18】 マイクロプロセッサによって制御される電子回路と、該マイクロプロセッサをターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムにおいて、
 前記電子回路、ターゲットマイクロプロセッサおよび制御プログラムは、所定のコンピュータ言語および所定のインタフェースライブラリを用いてシミュレートされ、
 前記コンピュータシステムは、
 前記電子回路の機能が所定のインタフェースライブラリの一部を含む所定のコンピュータ言語構文に翻訳されている前記電子回路のソフトウェアモデルと、
 前記制御プログラムの一部を実行するときに命令セットシミュレーションが前記ターゲットマイクロプロセッサによって実行される命令をシミュレートするような、前記ターゲットマイクロプロセッサの命令セットシミュレーションと、
 前記電子回路のソフトウェアモデルと、前記ターゲットマイクロプロセッサの命令セットシミュレーションを互いに結合する中央制御プロセスと、を有し、
 前記中央制御プロセスは、
 複数のグローバル信号の状態を管理する信号メンテナサブプロセスと、
 前記ソフトウェアモデルおよび命令セットシミュレータによって使用されるクロック信号を生成するクロック信号ジェネレータサブプロセスと、
 前記ソフトウェアモデルおよび命令セットシミュレータ

からのイベントをウェイトするプロセスキューを管理するキューメンテナサブプロセスと、
 所定のタイミング間隔を生成するタイマサブプロセスと、
 少なくとも前記信号メンテナサブプロセス、クロック信号ジェネレータサブプロセス、キューメンテナサブプロセスおよびタイマサブプロセスの実行を制御する制御サブプロセスと、を有することを特徴とするコンピュータシステム。

10 【請求項19】 前記所定のタイミング間隔は、1ナノ秒以上の時間間隔をシミュレートすることを特徴とする請求項18記載のコンピュータシステム。

【請求項20】 前記所定のタイミング間隔は、1ナノ秒未満の時間間隔をシミュレートすることを特徴とする請求項18記載のコンピュータシステム。

【請求項21】 前記複数のグローバル信号に対する状態変化を表示する信号ディスプレイサブプロセスをさらに有することを特徴とする請求項18記載のコンピュータシステム。

20 【請求項22】 電子回路と、該電子回路をターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムのための実行可能プログラムにおいて、

前記電子回路および制御プログラムは、所定のコンピュータ言語を用いてシミュレートされ、

前記実行可能プログラムは、

コンピュータ上で実行されるときに、複数のグローバル信号の状態を管理する第1実行可能コード部分と、

コンピュータ上で実行されるときに、複数のクロック信号を生成する第2実行可能コード部分と、

30 コンピュータ上で実行されるときに、イベントをウェイトするプロセスキューを管理する第3実行可能コード部分と、

コンピュータ上で実行されるときに、所定のタイミング間隔を生成する第4実行可能コード部分と、

コンピュータ上で実行されるときに、少なくとも前記第1、第2、第3および第4実行可能コード部分の実行を制御する第5実行可能コード部分と、

40 を有することを特徴とする、電子回路と制御プログラムとのコバリデーションを行うコンピュータシステムのための実行可能プログラム。

【請求項23】 電子回路と、該電子回路を制御する制御プログラムとを含むシステムの検証のための、ターゲットプロセッサのサイクル精度の命令セットシミュレーションを導出する方法において、

メモリオペレーションの完了を待機することによってキャッシュがロードされるまで前記ターゲットプロセッサがストールされるように、メモリとキャッシュの間の相互作用をデータバス上の信号のシーケンスとしてモデル化するステップと、

前記ターゲットプロセッサの内部データフローモデルから導出される信号のシーケンスを用いるとともに、内部バス幅およびタイミングを用いて、命令パイプラインを充填して、適当なクロックサイクル数だけ遅延を行うステップと、

前記ターゲットプロセッサの命令デコードサイクルを実行して、実行のために利用可能な命令を解釈するステップと、

スケジューリングされた命令がパイプライン切断をインプリメントしているかどうかを判定し、スケジューリングされた命令がパイプライン切断をインプリメントしている場合、将来の命令のためにスケジューラをストールさせるステップと、

スケジューリングされた命令を、適当な命令パイプラインまたはハードウェアコンポーネントに転送し、各命令ごとにサイクル実行時間を計算するステップと、非決定性タイミングがあるかどうかを判定するステップと、

計算された実行サイクルの終端で利用可能な結果を出力するステップと、

前記ターゲットプロセッサのエミュレートされた制御レジスタとともに、信号インタフェースを用いて、割込みハンドラが次のサイクルにスケジューリングされるべきかどうかを決定するステップと、

を有することを特徴とする命令セットシミュレーションを導出する方法。

【請求項 24】 前記非決定性タイミングがあるかどうかを判定するステップは、

非決定性タイミングがある場合、各クロックサイクルにおいて終了条件を評価するステップと、

非決定性タイミングがない場合、計算されたタイミングに対するリソース割当てを伝搬させるステップと、

を含むことを特徴とする請求項 23 記載の方法。

【請求項 25】 前記スケジューリングされた命令がパイプライン切断をインプリメントしているかどうかを判定するステップは、

スケジューリングされた命令がパイプライン切断をインプリメントしている場合、将来の命令のためにスケジューラをストールさせるステップを有することを特徴とする請求項 23 記載の方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、軽量のメッセージベースのオペレーティングシステムを用いてハードウェアおよびソフトウェアのコデザイン (co-design、協調設計) およびコバリデーション (co-validation、協調検証) を行う方法および装置に関する。特に、電気回路設計はソフトウェアプロセスにマッピングされ、マッピングされた電気回路要素間の状態変化は、軽量オペレーティングシステムのメッセージ受渡しプリミティブを用

いて渡される。軽量オペレーティングシステム下で動作する命令セットシミュレータにより、制御ソフトウェア要素と、マッピングされた電気回路要素とはコシミュレート (co-simulate、協調シミュレート) され、それらの相互作用の評価が可能となる。本発明は、ソフトウェアおよびハードウェアのコシミュレーション (co-simulation、協調シミュレーション) 環境を提供するコバリデーション方法、コシミュレーション方法をインプリメント (実装) するコンピュータシステム、コシミュレーション方法をインプリメントするソフトウェア命令を含むコンピュータプログラム製品、ならびに、ハードウェア要素のソフトウェアシミュレーションおよび命令セットシミュレーションをサポートするソフトウェア環境として、実現される。また、本発明は、ターゲット中央処理装置 (CPU) に対する命令セットシミュレータを開発する方法としても実現される。

【0002】

【従来の技術】以下の参照事項は、すべて本発明に関連する示した主題についての有用な背景的情報を提供する。

【0003】PTOLEMYは、並行 (コンカレント) システムの異種 (heterogeneous) モデリング、シミュレーションおよび設計を研究するプロジェクトである。現在の実装は、J A V A (登録商標) プログラミング言語で実現されている。<http://ptolemy.eecs.berkeley.edu>にあるPTOLEMYウェブサイトは、PTOLEMYに関するさらに詳細な情報を含む。

【0004】POLISは、組込みシステムのハードウェア/ソフトウェアコデザインのためのソフトウェアプログラムである。POLISは、PTOLEMYのために開発されたフレームワークを利用する。<http://www-cad.berkeley.edu/~polis/>にあるPOLISウェブサイトは、PTOLEMYに関するさらに詳細な情報を含む。

【0005】次に、本発明を理解するための適当な基礎を提供するいくつかの主題について記述する。

【0006】一般に、軽量オペレーティングシステムは、ハードウェアとアプリケーションレベルのコードとの間の多くの障壁を除去することによって、アプリケーションが高速に動作することを可能にする。1つの直接的な結果として、システムに残る保護機構は、あるとしてもわずかとなる。このため、このような環境は、組込みあるいは専用アプリケーションに適したものにはなるが、不完全に設計された、あるいは、悪意のある結果を生じるように意図されたアプリケーションに適したものとはならない。従来、このことは、ソフトまたはハードなリアルタイム組込みコードへの、マイクロカーネルシステムの利用を制限してきた。

【0007】現在、良好なハードウェアソフトウェア・コシミュレーションツールに対する需要が、次の3つ

の主要な要因によって引き起こされている。

1. コンピュータシステム（ハードウェアおよびソフトウェアの両方）のサイズおよび複雑さの増大。
2. 費用効果の高いSOC (system-on-a-chip)実装に対する要求。
3. これまでの投資に対する収益を最大化するようなIP (intellectual property)の再利用。

【0008】現在、一般に、上記の要因に対する有効な解決策は、ハードウェアおよびソフトウェアの両方のコンポーネントを必要とすることが認識されている。このことは、設計空間を「コデザイン」要求へと切り開いた。多くのタスクが、汎用の中央処理装置によっても、専用（あるいはプログラム可能）ハードウェアによっても実行可能である場合、最も有効な分割はどのようにしてなされるか。設計ステップは、次のようなフィードバックループで作用する。

1. タスクへの初期分割を行い、タスクアルゴリズムをコーディングする。
2. タスクをハードウェアまたはソフトウェアコンポーネントに割り当て、これらの割当てに従って適当な実行可能コードを生成する。
3. 高レベルシミュレーションを行い、基本機能および動作制約を確認する。
4. 設計基準を満たすように、エラーを訂正し可能な再分割を行う。
5. ハードウェアコンポーネントの完全な合成と、ソフトウェアの最適化を行う。
6. すべてのコンポーネントの低レベル（タイミングの正確な）シミュレーションを行う。
7. 許容可能なあるいは最適解に向けて反復する。

【0009】この点まで採用されたこの手順に対する代表的なアプローチが、PTOLEMY/POLIS環境で例証されている。PTOLEMY/POLISは、タスクがソフトウェアあるいはハードウェアのいずれの実装で合成されることも可能にすることによって、真のコデザイン環境であることを主張している。PTOLEMY/POLIS環境は、与えられた分割の検査を可能にする統合シミュレーションツールのためのフレームワークを提供するとともに、合成された結果を生成する最終的なコード生成フェーズを提供する。

【0010】しかし、PTOLEMY/POLIS環境のアーキテクチャは、その作者およびその一次ユーザ（すなわち、ハードウェアエンジニア）の先入観を反映している。PTOLEMY/POLIS環境には、システム設計がコシミュレートされるためのソフトウェアコンポーネントに対する最小限のサポートしかない。このことは、ソフトウェア実装のためのものを含めて、ESTERELにおけるすべてのタスク仕様を書く必要があることから明らかである。ESTERELは、状態マシン型のVHDLに対するフロントエンドとしては適当な

プログラミング言語であるが、ソフトウェアのためにCへと合成される言語としては大変である。

【0011】プログラマの観点から見ると、「良い」ESTERELであっても、「悪い」Cを生成する。実際、ESTERELは、そのプログラミング様式（単純なif-thenテストと、goto分岐）において、FORTRANに非常に近いコードを生成する。このことは、ある程度のマシンコードを生成するだけでもCコンパイラに重い負荷をかけ、よりグローバルな最適化を行う能力を制限する。また、この条件のため、ソフトウェアエンジニアは、既存のIPを捨てることを余儀なくされ、有用で試験済みの（しかも信頼された）アルゴリズムを再コーディングしなければならない。オブジェクト指向技術のサポートがないことはいかに及ばず、最新の言語の高水準の特徴がないため、ソフトウェア分割は、この階層における下層階級となる。

【0012】最新のプログラミング言語（例えば、C++）および高水準ハードウェア設計言語の両方の表現力を有する言語がないため、統一されたコデザイン環境に取り組みようとするのにさえかなりの抵抗がある。

【0013】PTOLEMYフレームワークの別の実現も可能であり、以上の問題点の解決に役立つ可能性がある。このようなシステムは、ソースコード形式で入手不可能なこともある市販のIPを含む既存のコードを利用することができなければ、可能な設計空間を制限し、最適な結果を排除することになる可能性がある。（コストや市場出荷までの期間(time-to-market)の理由から）主な問題点が、ハードウェアとソフトウェアにすでに分割された既存の「レガシー」IPの再利用である場合、設計空間は変わる。例えばCコードやVHDLのような複数の方式で存在するコンポーネントが容易に思いつく。この場合、問題は次のようになる。

1. 一緒に用いられる場合に設計制約を満たす既存のIPのセットが存在するか。
2. 存在する場合、制御信号およびデータが正しく流れるように、これらのプロセスをどのように「のり付け」しなければならないか。
3. 存在しない場合、まずい点はどこか。また、その点を解決するのに必要なアプローチは何か。

【0014】これらの問題は、既存の設計どうしの間のインタフェースと、システム全体の総合的な正しい動作とを重要視し、したがって、コシミュレーションおよびコベリフィケーション(co-verification)の点を重要視したものである。

【0015】動的再分割のことを仮に除いて考えると、プログラマは、自分の必要に最も適した言語（1つの言語または複数の言語）で自分の考えを表現する自由がある。追加の「のり(glue)」は、ハードウェアエンティティ（例えば、アドレスデコーディングロジック）やソフトウェアモジュール（例えば、CPU外ハードウェアアチ

ップのためのドライバ)として提供可能である。この段階で必要とされるのは、設計を評価するための迅速なフィードバックシステムである。

【0016】ここでも、市販のコシミュレーションツールは、約束するもののすべてを提供してはいない。ときとしては、この「ツール」は、市販のシミュレータを接続することができる単なる統合フレームワークである。これは、多くの重要な作業を節約し、これが「プラグアンドプレイ」フレームワークを提供する場合には、それぞれの要求を解決するのに最適なツールが接続されることが可能となる。しかし、個々のコンポーネントどうしの間に直接の通信がなく、したがって直接のフィードバックがない場合には、設計空間が増大すると、このアプローチは最終的に機能しなくなる可能性がある。このような実装は、現在、あるツールから別のツールにトレースを供給して収束を待つ反復的「逐次近似」法によって特徴づけられる。

【0017】1つの代替アプローチは、要求されるフィードバックパスをすでに備えたアーキテクチャフレームワークおよびコシミュレーション環境を開発し、これに、一連の問題群全体にわたり役立つ可能性のあるターゲット変更可能(re-targetable)ツールを配置するものである。

【0018】ハードウェアエンジニアリングの観点ではなくソフトウェアの観点からコシミュレーション問題を検討し、特に、SOC集積の同様のアプリケーションを考慮すると、次のようないくつかの類似点が浮かび上がる。

【0019】1 a. 軽量ソフトウェアシステムは、小規模の(ときにはリアルタイムの)スケジューラを通る相互通信プロセスのセットとして書かれる傾向がある。

1 b. VHDLコードは、ワイヤのセットを通る相互通信エンティティのセットとして書かれる傾向がある。

【0020】2 a. ソフトウェアプロセスはイベント駆動型である傾向がある。すなわち、ソフトウェアプロセスは、何らかの他のプロセスまたは外部信号をウェイト(待機)し、何らかの処理を実行し、そして、次のイベントをウェイトする。

2 b. ハードウェアエンティティはイベント駆動型である傾向がある。すなわち、ハードウェアエンティティは、入力に現れる何らかの条件をウェイトし、何らかの処理を実行し、そして、次のイベントをウェイトする。

【0021】3 a. ソフトウェアプロセス間のイベントは通常、送信側が情報の「パケット」(データ領域)を用意し、データ同期ポイントとして作用するスケジューラを通して、「信号」を別のプロセスに送ることによって進行する。

3 b. ハードウェアプロセス間のイベントは通常、送信側が「データ」ワイヤのセット(例えば、バス)上に値を用意し、データ同期ポイントとして作用するグルーロ

ジック(例えば、ラッチ)を通して、信号(例えば、チップ選択(chip-select)信号)を別のチップに送ることによって進行する。

【0022】4 a. シングルCPUシステムでは、真のソフトウェア並列化はなく、見かけの並列化が、オペレーティングシステムがタスク間を切り換えることによって行われる。

4 b. きわめて自明なハードウェアを除くすべてのハードウェアでは、真の並列化があり、これは、独立のハードウェアユニットが分散クロックを受信し、それぞれそのそのクロックサイクル中に自己のタスクを実行することによって達成される。

【0023】5 a. 適切に設計されたソフトウェアシステムでは、個々のプロセスは、あらかじめ定義されたプロセス間通信経路(これは、動的に割り当てられることも可能である)を除いては、独立に動作する。

5 b. 適切に設計されたハードウェアシステムでは、ハードウェアプロセスの個々のインスタンスは、信号およびワイヤを用いて適切に定義された通信経路(これは、「グルーロジック」内の設定によって決定されることも可能である)を除いては、単独で動作する。

【0024】VHDLコードのシミュレーションのためのツール(例えば、Model Technologies社のVSIM)がすでに存在する。このようなツールは、システム全体の内部の個々のエンティティの挙動をエミュレートすることによって動作し、上記の項目5 bに基づいて、ハードウェア並列化の代わりにソフトウェア型の並列化(すなわち、順次実行)を行う。これは、同期ポイントにおける個々のプロセスの状態(例えば、デルタあるいはクロック遷移)をそれぞれチェックし、各シミュレーションサイクルの終端で状態変化を行うことによってなされなければならない。通常、シミュレーション環境全体が、ワークステーション上の単一プロセスとして動作し、内部スケジューラが、その内部のVHDLプロセスの状態を追跡する。

【0025】図1に、従来のVHDLシミュレーションを例示する。通常、シミュレーション環境は、マルチプロセスオペレーティングシステム(例えばUNIX(登録商標)1)内で実行される。シミュレーションのコンポーネントは、命令セットシミュレータ4、VHDLシミュレーション5、および、信号を受渡しするためのインタフェース6である。VSIM環境2は、内部スケジューラ3を用いて、シミュレーションコンポーネントを制御する。図1に示されているように、割込み7は、VHDLシミュレーション5から、インタフェースを通じて、命令セットシミュレータに渡されなければならない。外部コンポーネント8との相互作用は、内部スケジューラ3を通してVSIM環境2に渡されなければならない。さらに、VSIMシミュレータ2は、UNIXシステム1を通じてデータを渡さなければならない。オ

ペレーティングシステムにとってV S I M環境2の全体がU N I Xオペレーティングシステムシェル1で実行される単一のプロセスとして見えるため、内部スケジューラ3は、従来のシミュレーションシステムの本質的コンポーネントである。同時に、このプロセスが、システムの他の外部コンポーネント8と相互作用することを可能にするため、U N I Xオペレーティングシステム1のマルチプロセッシングサポートが要求される。

【0026】現実のシステムは、単純な「部分の総和」アプローチが示すようなものよりもしばしばはるかに複雑であり、システム統合の問題は、個々の設計フェーズ中に見落とされる設計欠陥を明らかにすることがある。このことは特に、非決定性システム（例えば、ベストエフォート型ネットワークリンクを通じて受信されるデータの処理）の設計の際には明らかである。また、設計空間は、起こりうるさまざまなエラー（パケット損失、データ破損、リアルタイムデッドライン超過など）と、それが実際に起きたときにエラーを処理するストラテジとを考慮に入れなければならない。

【0027】実世界のアプリケーションにおけるもう1つの問題点は、データの周期性が、マイクロ秒やナノ秒ではなく、数分の1秒以上となることが多いことである。例として、M P E Gデータの表示システムを考える。（毎秒30フレームの表示の場合）33msという基本フレーム周期があるが、これは、データの真の周期ではない。M P E Gは、データの（単一の）フルフレームと、それを修正する一連の補間フレームとを有するG O P (group of pictures)セグメントへと、フレームをクラスタ化する。通常、G O Pは、15フレーム、すなわち、0.5秒のオーダーのデータである。フルフレームデータの損失は実質的にそのG O P内の後続フレームを無意味にするため、このレイヤはエラーの伝搬を制限する。

【0028】したがって、このようなシステムの設計においては、1個のG O P全体のデータより短いシミュレーションは、不正なエラー処理状況を見落とす可能性が高い。実際、秒のオーダーのデータと、1分に達するシミュレートされた実行を要求しない試験ストラテジを考えることは困難である。このようなシステムでは、任意の設計に対して、少なくとも2つの「予備」ステップがあることになる。第1のステップは、その設計がフルスピードM P E Gデータのリアルタイム制約を処理することの確認である。第2のステップは、ずっと長い期間にわたるエラー条件をどの程度うまく処理するかについて調べることである。

【0029】

【発明が解決しようとする課題】より強力な（例えば、コスト、エネルギーに関して）C P Uと、追加のカスタムハードウェア（例えば、F P G AやA S I C）の間には一般的なトレードオフがある。上記のM P E Gの例を

用いると、M I P Sプロセッサでハードウェア支援がない場合、10フレーム/秒を超える表示は不可能である。しかし、インテルMMXレベルのプロセッサの場合、追加ハードウェアがなくても、30フレームを優に超える表示が可能である。このような観察は、ハードウェアおよびソフトウェアの両方に対して単一の設計言語を使用することに対する強力な反論となる。というのも、このパフォーマンスは、主要なルーチンをインテルのアセンブリ言語で直接にコーディングすることによってのみ達成することができるからである。当面する問題を正しく解決することが可能な機能を無視することによって、候補となる可能性のあるC P Uを「不自由にする」のはほとんど無意味である。

【0030】

【課題を解決するための手段】本発明は、上記の状況に鑑み、従来技術の上記の問題点と制限を克服するものである。

【0031】本発明のさらに他の利点については、その一部は以下の説明に記載されており、一部はその説明から明らかであるか、または、本発明を実施することにより知ることができる。本発明の利点は、特許請求の範囲に特に記載された手段と組合せによって実現され達成される。

【0032】本発明の第1の特徴によれば、電子回路と、その電子回路をターゲットとする制御プログラムとのコバリデーション（協調検証）を行う方法が実現される。ここで、電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされる。この方法は、軽量コンピュータシステムオペレーティング環境をターゲットとする構文(construct)からなるハードウェアモデルに、電子回路の動作シミュレーションを翻訳することを含む。さらに、この方法は、サイクル精度の命令セットシミュレーションを構成することを含む。これにより、命令セットシミュレーションは、電子回路をターゲットとする制御プログラムの一部を実行する。さらに、この方法は、電子回路のハードウェアモデルを命令セットシミュレーションに結合して、軽量コンピュータオペレーティングシステム環境で実行される試験対象システムを作成することを含む。さらに、この方法は、試験対象システムに刺激を入力することにより、試験対象システムに結果を出力させることを含む。

【0033】本発明の第2の特徴によれば、電子回路と、その電子回路をターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムが実現される。ここで、電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされる。このコンピュータシステムは、すべてのグローバル信号の状態を管理するソフトウェア手段を有する。さら

に、このコンピュータシステムは、クロック信号を生成するソフトウェア手段を有する。さらに、このコンピュータシステムは、イベントをウェイトするプロセスキューを管理するソフトウェア手段を有する。さらに、このコンピュータシステムは、所定のタイミング間隔で少なくとも1つのプロセスを実行するソフトウェア手段を有する。さらに、このコンピュータシステムは、中央プロセス制御を行うソフトウェア手段を有する。

【0034】本発明の第3の特徴によれば、電子回路と、その電子回路をターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムが実現される。ここで、電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされる。このコンピュータシステムは、プロセッサと、コンピュータシステムが中央制御プロセスを実行することを可能にするように適応したソフトウェア命令を含むメモリとを有する。中央制御プロセスは、複数のグローバル信号の状態を管理する信号メンテナサブプロセスと、ソフトウェアモデルおよび命令セットシミュレータによって使用されるクロック信号を生成するクロック信号ジェネレータサブプロセスとを有する。さらに、中央制御サブプロセスは、ソフトウェアモデルおよび命令セットシミュレータからのイベントをウェイトするプロセスキューを管理するキューメンテナサブプロセスと、所定のタイミング間隔を生成するタイマサブプロセスとを有する。さらに、中央制御サブプロセスは、少なくとも信号メンテナサブプロセス、クロック信号ジェネレータサブプロセス、キューメンテナサブプロセスおよびタイマサブプロセスの実行を制御するコントローラサブプロセスを有する。

【0035】本発明の第4の特徴によれば、マイクロプロセッサによって制御される電子回路と、そのマイクロプロセッサをターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムが実現される。ここで、電子回路、ターゲットマイクロプロセッサおよび制御プログラムは、所定のコンピュータ言語および所定のインタフェースライブラリを用いてシミュレートされる。このコンピュータシステムは、電子回路のソフトウェアモデルを有する。電子回路の機能は、所定のインタフェースライブラリの一部を含む所定のコンピュータ言語構文に翻訳されている。さらに、このコンピュータシステムは、ターゲットマイクロプロセッサの命令セットシミュレーションを有し、これにより、命令セットシミュレーションは、制御プログラムの一部を実行するときにターゲットマイクロプロセッサによって実行される命令をシミュレートする。さらに、このコンピュータシステムは、電子回路のソフトウェアモデルと、ターゲットマイクロプロセッサの命令セットシミュレーションを互いに結合する中央制御プロセスを有する。中央制

御プロセスは、複数のグローバル信号の状態を管理する信号メンテナサブプロセスと、ソフトウェアモデルおよび命令セットシミュレータによって使用されるクロック信号を生成するクロック信号ジェネレータサブプロセスとを有する。さらに、中央制御サブプロセスは、ソフトウェアモデルおよび命令セットシミュレータからのイベントをウェイトするプロセスキューを管理するキューメンテナサブプロセスと、所定のタイミング間隔を生成するタイマサブプロセスとを有する。さらに、中央制御サブプロセスは、少なくとも信号メンテナサブプロセス、クロック信号ジェネレータサブプロセス、キューメンテナサブプロセスおよびタイマサブプロセスの実行を制御する制御サブプロセスを有する。

【0036】本発明の第5の特徴によれば、電子回路と、その電子回路をターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムのための実行可能プログラムが実現される。ここで、電子回路および制御プログラムは、所定のコンピュータ言語を用いてシミュレートされる。この実行可能プログラムは、複数のグローバル信号の状態を管理する第1実行可能コード部分を有する。さらに、この実行可能プログラムは、複数のクロック信号を生成する第2実行可能コード部分を有する。さらに、この実行可能プログラムは、イベントをウェイトするプロセスキューを管理する第3実行可能コード部分を有する。さらに、この実行可能プログラムは、所定のタイミング間隔を生成する第4実行可能コード部分を有する。さらに、この実行可能プログラムは、少なくとも第1、第2、第3および第4実行可能コード部分の実行を制御する第5実行可能コード部分を有する。

【0037】本発明の第6の特徴によれば、電子回路とその電子回路を制御する制御プログラムとを含むシステムの検証のための、ターゲットプロセッサのサイクル精度の命令セットシミュレーションを導出する方法が実現される。この方法は、メモリとキャッシュの間の相互作用をデータバス上の信号のシーケンスとしてモデル化することを含む。ここで、ターゲットプロセッサは、メモリオペレーションの完了を待機することによってキャッシュがロードされるまでストールされる。さらに、この方法は、ターゲットプロセッサの内部データフローモデルから導出される信号のシーケンスを用いるとともに、内部バス幅およびタイミングを用いて、命令パイプラインを充填して、適当なクロックサイクル数だけ遅延を行うことを含む。さらに、この方法は、ターゲットプロセッサの命令デコードサイクルを実行して、実行のために利用可能な命令を解釈することを含む。さらに、この方法は、スケジューリングされた命令がパイプライン切断をインプリメントしているかどうかを判定し、スケジューリングされた命令がパイプライン切断をインプリメントしている場合、将来の命令のためにスケジューラをス

ツールさせることを含む。さらに、この方法は、スケジューリングされた命令を、適当な命令パイプラインまたはハードウェアコンポーネントに転送し、各命令ごとにサイクル実行時間を計算することを含む。さらに、この方法は、非決定性タイミングがあるかどうかを判定する。さらに、この方法は、計算された実行サイクルの終端で利用可能な結果を出力することを含む。さらに、この方法は、ターゲットプロセッサのエミュレートされた制御レジスタとともに、信号インタフェースを用いて、割込みハンドラが次のサイクルにスケジューリングされるべきかどうかを決定することを含む。

【0038】本発明の上記の特徴およびその他の利点は、以下の詳細な説明から、また、添付図面を参照して、明らかとなる。

【0039】

【発明の実施の形態】本発明の特徴について説明する前に、本発明の理解を助けるため、および、さまざまな用語の意味を提示するために、従来技術に関していくつかの詳細な事項について説明する。

【0040】本明細書において、「コンピュータシステム」という用語は、可能な限り最も広い意味を含み、独立（スタンドアロン）のプロセッサ、ネットワーク化されたプロセッサ、メインフレームプロセッサ、およびクライアント／サーバ関係にあるプロセッサを含むが、これらに限定されない。「コンピュータシステム」という用語は、少なくともメモリおよびプロセッサを含むものと理解されるべきである。一般に、メモリは、いろいろなときに、実行可能プログラムコードの少なくとも一部を格納し、プロセッサは、その実行可能プログラムコードを構成する命令を実行する。

【0041】本明細書において、「組込みコンピュータシステム」という用語は、組込み中央プロセッサとオブジェクトコード命令を有するメモリとからなるが、これに限定されない。組込みコンピュータシステムの例には、携帯情報端末（PDA）、セルラ電話機およびデジタルカメラがあるが、これらに限定されない。一般に、どんなに原始的であっても、中央プロセッサを用いてその機能を制御する機器は、組込みコンピュータシステムを有するということができる。組込み中央プロセッサは、メモリに格納されたオブジェクトコード命令を実行する。組込みコンピュータシステムは、キャッシュメモリ、入出力デバイスおよびその他の周辺装置を有することが可能である。

【0042】理解されるように、「所定オペレーション」という用語および「コンピュータシステムソフトウェア」という用語は、本明細書の目的では、実質的に同じものを意味する。本発明の実施にとって、メモリおよびプロセッサが物理的に同じ場所に位置する必要はない。すなわち、プロセッサおよびメモリが物理的に異なる装置にあることや、地理的に異なる場所にあることも

予期される。

【0043】本明細書において、当業者には理解されるように、「媒体」あるいは「コンピュータ可読媒体」は、ディスケット、テープ、コンパクトディスク、集積回路、カートリッジ、通信回線を通じてのリモート伝送、あるいはその他の同様の、コンピュータが使用可能な媒体を含むことが可能である。例えば、コンピュータシステムソフトウェアを分散させるため、供給業者（サプライヤ）は、ディスケットを提供することも可能であり、また、衛星伝送、直接電話リンク、あるいはインターネットを通じて何らかの形で所定オペレーションを実行する命令を伝送することも可能である。

【0044】コンピュータシステムソフトウェアは、ディスケットに「書き込まれる」ことも、集積回路に「格納される」ことも、通信回線を通じて「伝送される」ことも可能であるが、理解されるように、本明細書の目的では、コンピュータ使用可能媒体は、所定オペレーションを実行する命令を「有する」ということにする。すなわち、「有する」という用語は、所定オペレーションを実行するための命令がコンピュータ使用可能媒体に関わる上記およびすべての等価な方法を含むことを意図するものである。

【0045】したがって、簡単のため、「プログラム製品」という用語は、以下、任意の形式で所定オペレーションを実行するための命令を有する（上記で定義）コンピュータ使用可能媒体を指すために用いられる。

【0046】次に、添付図面を参照して、本発明の特徴の詳細な説明を行う。

【0047】VHDLプロセスと、軽量オペレーティングシステムで実行されるプロセスとの間の顕著な類似性のため、VHDLプロセスを、個別のソフトウェアプロセスとして、軽量オペレーティングシステムを用いて書き、イベントをスケジューリングすることが可能となる。好ましくは、軽量オペレーティングシステムは、メッセージによるプロセス間通信を提供する小さいマイクロカーネルをもとに構築される。メッセージは、関連するパラメータを有するオペレーションコードであり、上記の項目3aのイベント駆動モデルを表現する。好ましくは、メッセージは、プロセス間で受渡しされるシステム構造体にマッピングされるグローバルデータ定義である。イベントのセマンティクスはアプリケーションによって定義される。この場合、プロセス間メッセージは、VHDL信号値の変化についてプロセスに通知するために用いられることも可能である。好ましくは、軽量オペレーティングシステムは、信号線、アドレスバス、データバスおよびクロック分配方式が少数の基本メッセージから構築されることを可能にする一群のメッセージプリミティブを提供する。好ましくは、マイクロカーネルコアによって供給されるメッセージに加えて、プロセスは、アプリケーション固有のインタフェースをインプリ

メントするために自己のメッセージコードおよびパラメータを定義することも可能である。その場合、このインタフェースは、共有ライブラリにインプリメントされたルーチンコールのセットにまとめられる。

【0048】次に、図2を参照して、電子回路と、その回路をターゲットとするソフトウェアとのコバリデーションを行う本発明の特徴について簡単に説明する。電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされる。S0900で、1つまたは複数の回路記述言語によってモデル化された、コンピュータハードウェアの動作の要素を、選択された軽量オペレーティングシステムの構文およびメソッドに翻訳する。S1000で、電子回路設計の動作記述を、軽量コンピュータシステムオペレーティング環境をターゲットとする前記構文からなるソフトウェアモデルに翻訳する。S1100で、電子回路設計をターゲットとする制御プログラムの部分を命令セットシミュレータを所定のコンピュータ言語で構成する。S1200で、電子回路設計のソフトウェアモデルを命令セットシミュレーションに結合して、軽量コンピュータオペレーティングシステム環境で実行される試験対象システムを作成する。最後に、S1300で、試験対象システムに刺激を入力することにより、試験対象システムに結果を出力させる。

【0049】次に、電子回路および制御プログラムのコバリデーションを行う本発明の特徴についてさらに詳細に説明する。

【0050】ステップS0900では、ハードウェア記述言語を調べる。好ましくはVHDL言語がモデル化されるが、本発明の方法および応用は、特定の言語や特定の軽量オペレーティングシステムに限定されない。動作レベルVHDLの制御フローおよび様式は、以下の具体的な点でCに容易にマッピングすることが可能である。

【0051】VHDLの変数はCの変数に似ている。その値は、制御がプロセスを離れて戻ってきたときにも保持されている。しかし、その名前前のスコープは、それがインスタンス化されたプロセスについてローカルである。したがって、本発明は、VHDLの変数をCのスタティック変数で置き換える。ビットベクタ型は、32ビット幅（32ビットプロセッサで動作する処理系の場合）までは、Cの符号なし整数型に含めることができる。個別ビットアクセスおよびスライス演算はいずれも、シフトおよびマスク演算で実行することができる。

【0052】信号は、プロセス間で値を送信するために用いられる。信号の値のスコープは、システム全体を通じてグローバルであるが、その名前は、特定のプロセスについてローカルである。本発明では、プロセスの複数のコピーを、それぞれ相異なる信号（例えば、UART_A_Chip_Select、UART_B_Chip_Select）にアクセスする同一

のローカル信号名（例えば、Chip_Select）でインスタンス化することが可能である。「0」や「1」のような単純な信号値と、スライス値とは、マシン内のビットパターンとして容易に表現される。VHDLは、用いられる電氣的モデルに依存して、「Z」や「W」のようなこの範囲外の論理値をサポートする。

【0053】個々のVHDLプロセスは、一定のナノ秒数の間、または、その信号に関する条件のセットが満たされるまで、その実行をサスペンドすることが可能である。VHDLプロセスが「動作可能」であるための条件を決定することは、言語構文のマッピングにおいてインプリメントされなければならない機能の1つである。

【0054】また、VHDL言語は、ハードウェアの実行のフローを制御する構文を提供する。これは、軽量オペレーティングシステムによって用いられるプログラミング言語（この場合はC）の等価な構文にマッピングされなければならない。

【0055】好ましくは、シミュレーションシステム内の中央プロセスは、各信号の値を追跡すること、および、その値をモニタするプロセス間に信号変化を分配することの処理を行う。以下の説明でこのプロセスに言及するときには、「ファブリック」プロセスと呼ぶことにする。本発明は、軽量オペレーティングシステムのメッセージ受渡しオペレーションを用いて、プロセス間で、信号値の変化やその他の構文の受渡しを行う。例えば、"wait for N ns"（Nナノ秒ウェイト）というVHDL構文は、次のような2つのユーザ定義パラメータを有するメッセージにより表現される。

```
Msg WAITFOR lint CLOCKT; int TICKS;
```

TICKSパラメータは、VHDLにおけるNパラメータからナノ秒数を提供し、CLOCKTパラメータは、クロック相対同期ポイントを提供する。

【0056】メッセージインタフェースは、アプリケーションコードに直接には現れない。すべての相互作用は、マイクロカーネルコア機能呼び出す共有ライブラリルーチンを通じてなされる。

【0057】信号は文字列を用いて命名されるが、opaque型ポインタを用いて参照される。以下のインタフェースルーチンは、信号名と参照の間のマッピングをインプリメントする。さらに、このインタフェースは、信号値の変化について中央制御プロセッサに通知し、信号の値の変化をウェイトする、共有ライブラリルーチンを含む。

【0058】・void *fabinit(void): このルーチンは、中央制御プロセスインタフェースライブラリを初期化する。このルーチンは、信号メンテナを使用することになる各プロセスによって1回呼び出されなければならない。

【0059】・void *fabric_find_signal(char *SIGNAL): このルーチンは、（グローバルに命名された）S

GNALパラメータを参照するために使用可能なopaque型ポインタを返す。このルーチンは、すべての登録された名前に対する文字列比較を実行し、信号参照が存在すればそれを返す。信号がリスト内にまだない場合、初期値0で作成される。このようにして、プロセスは、特定の順序でインスタンス化あるいは初期化されることは不要である。

【0060】・uint fabric_read_signal(void *SYM) :

このルーチンは、opaque型SYMパラメータによって参照される信号の現在値を、もしそれが定義された(すなわち、非浮遊トライステート) 値を有していれば、返す。

【0061】・void fabric_set_clock(char *NAME, int INITIAL, int PERIOD) : このルーチンは、このプロセスに対する同期オペレーションで用いられるクロック信号を定義する。このルーチンは、ファブリックプロセス内への共有ライブラリ呼出しであり、ファブリックプロセスがすべての遷移を内部的に処理する。INITIALパラメータは、信号の開始値(通常0)を指定し、PERIODパラメータは、遷移間の間隔(すなわち、クロック周期の半分)を質する。これは信号を定義するため、クロックはグローバルであり、複数のプロセスが同じクロックを共有することができる。

【0062】・void fabric_tristate(void *SYM, int VALUE) : このルーチンは、IEEE論理システムの拡張値の設定およびそれとの比較を可能にする多値論理信号をサポートする。

【0063】・void fabric_write_signal(void *SYM, uint VALUE) : このルーチンは、opaque型ポインタSYMによって参照される信号の値をVALUEにセットする。

【0064】ファブリックプロセスは、信号が、単純な2値ドメイン内で作用しているか、それとも、多値論理を使用しているかを知らなければならない。このドメインは、システムにわたり信号変化を送信するために用いられるインタフェースルーチンによって決定され、処理系は、多値論理ドメインが表現されることを可能にする。それぞれの多値論理ドメインに要求される拡張値の表現を含む個々のCヘッダファイル(例えば、ieee.h)は、個々の論理ドメインをカプセル化することが可能である(例えば、std_logicパッケージ)。

【0065】以下の形のWaitルーチンはすべて、CLOCKTパラメータに次の3つの値のうちの1つをとる。0は、非同期オペレーションを意味する(指定された条件が満たされるとすぐにプロセスをスケジューリングする)。-1は、条件がプロセスのクロック信号の立下り端で満たされる場合にプロセスをスケジューリングし、+1は、条件がプロセスのクロック信号の立上り端で満たされる場合にプロセスをスケジューリングする。

【0066】・void WaitCk(int CLOCKT) : このルーチンは、(CLOCKTパラメータに従って) そのクロックの

次の立上りまたは立下り端まで、現在のプロセスをサスペンドする。

・void WaitFor(uint TICKS, int CLOCKT) : このルーチンは、TICKS個のクロックティックの間、現在のプロセスをサスペンドする。

・void WaitUntilc(void *SYM, int OP, uint VALUE, int CLOCKT)

・void WaitUntil2c(void *SYM1, int OP1, uint VALUE1, void *SYM2, int OP2, uint VALUE2, int CLOCKT)

・void WaitUntil3c(void *SYM1, int OP1, uint VALUE1, void *SYM2, int OP2, uint VALUE2, void *SYM3, int OP3, uint VALUE3, int CLOCKT) : これらのルーチンは、1、2または3個の条件のうちのいずれか1つ以上が(同時に)真になるのをウェイトする。それぞれの条件は、信号SYM、オペレーションOPおよびVALUEである。例えば、"EQ"オペレーションは、SIGNALとVALUEの間の等値性をテストするために用いられる。

【0067】上記のルーチンは、プロセス間メッセージを形成し、それをファブリックプロセスに送る。このコントローラプロセスは、信号に対するすべての変化を追跡し、イベントをウェイトしているプロセスをいつ呼び起こすかを決定する。

【0068】これらのルーチンは、共有ライブラリにインプリメントされ、Cヘッダファイルを通じてエクスポートされた。また、ヘッダファイルは、ファブリックプロセスによって管理されるグローバルナノ秒ティックカウンタもエクスポートするため、個々のプロセスからのトレース出力は、現在のタイムスタンプでタグ付けされることが可能である。

【0069】図3を参照すると、本発明は、それぞれの信号名を前処理する。S1010で、それぞれの信号に対して、本発明は、名前にプレフィクス句"signal_"を付けることによって、内部変数を生成する。したがって、信号CHIP_SELECTへのopaque型ポインタは"static void *signal_CHIP_SELECT"となる。その後、この新しい信号名は、プロセスがインスタンス化されるときに、信号参照に結合される。インスタンス化された信号名がその信号名と同一である必要はない。例えば、signal_CHIP_SELECTは次のようにインスタンス化されることも可能である。

signal_CHIP_SELECT=fabric_find_signal("UART_A_CHIP_SELECT");

VHDLの本体で、プリプロセッサは、信号CHIP_SELECTへの参照を、変数signal_CHIP_SELECTを参照するように修正する。インスタンス化されるとき、変数signal_CHIP_SELECTは、UART_A_CHIP_SELECT信号に結合される。

【0070】S1015で、前処理すべき信号名がさるにあるかどうかを判定する。ある場合、S1018で、前処理すべき次の信号名を取得し、プロセス制御フローはS1010に戻って、信号名の前処理を続ける。ない

場合、プロセス制御フローはS1020に進む。

【0071】図3を参照して、プロセス制御フローがS1020～S1060に到達すると、どのタイプの動作言語構文が用いられているか、および、その動作言語を、軽量コンピュータオペレーティングシステムをターゲットとする構文にどのように翻訳するかについて判定を行う。好ましくは、使用される動作言語はVHDLであり、翻訳のために用いられる構文はCベースのものである。

【0072】S1020で、翻訳されるべき動作言語構文が内部変数への代入であるかどうかを判定する。内部変数への代入である場合、プロセス制御フローはS1025に進む。内部変数への代入でない場合、プロセス制御フローはS1030に進む。

【0073】S1025で、VHDL代入からC代入への翻訳が次のように行われる。

VHDL: i_word := 1;

C: static uint i_word;

i_word = 1;

この翻訳の終了後、プロセス制御フローはS1020に戻る。

【0074】S1030で、翻訳されるべき動作言語構文が外部変数への代入であるかどうかを判定する。外部変数への代入である場合、プロセス制御フローはS1035に進む。外部変数への代入でない場合、プロセス制御フローはS1040に進む。

【0075】ほとんどのインタフェースルーチンは、VHDL-C翻訳が読みやすくかつ理解しやすくなるように設計されたプリプロセッサマクロを通じてアクセスされる。これらのマクロは、信号名と変数名の間の変換を処理する。以下のマクロにおいて、“signal”はVHDL信号の名前を意味する。これらの名前は、上記のopaque型参照へと前処理される。

・uint GetSignal(signal NAME): このマクロは、NAMEという信号の現在値を返す。

・void Signal(signal NAME, uint VALUE): このマクロは、NAMEという信号のVALUEをその(32ビットの)値にセットする。

・void SignalZ(signal NAME): このマクロは、NAMEという信号の値をIEEEのZ値にセットする。

【0076】S1035で、VHDL代入から外部信号への翻訳が次のように行われる。

VHDL: a_busp <= '0';

マクロ: Signal(a_busp, 0);

C: fabric_write_signal(signal_a_busp, 0);

生成される代入は、ランタイムインタフェースを通じてファブリックプロセスへの呼出しを生成することになる。この翻訳の終了後、プロセス制御フローはS1020に戻る。

【0077】S1040で、翻訳されるべき動作言語構

文がウェイトポイントであるかどうかを判定する。ウェイトポイントである場合、プロセス制御フローはS1045に進む。外部変数への代入でない場合、プロセス制御フローはS1050に進む。外部信号への代入と同様に、ウェイトポイントは、次のようなプリプロセッサマクロを用いたインタフェース呼出しにマッピングされる。

・void WaitClk |FIR| (void): これらのマクロはそれぞれ、プロセスクロック信号の次の立上りまたは立下り端をウェイトする。

・void WaitFor |FIR| (uint TICKS): これらのマクロは、TICKSナノ秒間ウェイトする。プロセスは、そのティックにおける、または、次の適当なクロック遷移における実行のためにスケジューリングされる。

・void WaitUntil |FIR| (signal NAME, int OP, uint VALUE)

・void WaitUntil2 |FIR| (signal NAME, int OP, uint VALUE, signal NAME2, int OP2, uint VALUE2)

・void WaitUntil3 |FIR| (signal NAME, int OP, uint VALUE, signal NAME2, int OP2, uint VALUE2, signal NAME3, int OP3, uint VALUE3): これらのマクロ

は、ある条件が1、2、または3個の信号に現れるまでプロセスがウェイトすること、および、要求されるクロック同期に従ってプロセスがスケジューリングされることを可能にする。

【0078】S1045で、VHDL構文の翻訳は次のように行われる。

VHDL: wait until (startn = '0');

マクロ: WaitUntil(startn, EQ, 0);

C: WaitUntilc(signal_startn, EQ, 0, 0);

ウェイトポイントの翻訳の終了後、プロセス制御フローはS1020に戻る。

【0079】S1050で、VHDL制御構文が、等価なC言語オペレーションに翻訳される。この翻訳は、追加のインタフェース呼出しを含むことがある。例として、'if'制御構文の翻訳は次のように行われる。

VHDL: if (resetn = '0') then

マクロ: if (GetSignal(resetn) == 0) |

C: if (fabric_read_signal(signal_resetn) == 0)

|

制御構文の翻訳の終了後、プロセス制御フローはS1020に戻る。

【0080】図4を参照すると、S1060で、翻訳する必要のある動作言語構文がさらにあるかどうかを判定する。さらにある場合、プロセス制御フローはS1020に進む。さらでない場合、変換プロセスは完了する。

【0081】これらの少数の基本変換により、動作VHDLコードをCに翻訳し、そのコードをコンパイルし、軽量オペレーティングシステム用にそれをリンクすることが可能である。CとVHDLの間のマッピングは一對

一であり可逆であるため、もとの設計がVHDLで定式化されているか、それとも、翻訳可能Cで定式化されているかで、シミュレーションには差がない。翻訳が正しく実行される限り、結果は同じになる。この翻訳から生じるCのサブセットは依然として強力なプログラミング言語である。これは、ポインタ間接参照や高度なデータ構造体のような特にCPUベースのアルゴリズムを目的とするいくつかの機能を欠いているが、それ以外の点では、ターゲット変更可能設計の出発点として働くことが可能である。分割がわかったら、個々の解（ハードウェアまたはソフトウェア）の機能を十分に活用すべきである。

【0082】本発明の特徴によれば、設計のソフトウェアコンポーネントは、ターゲットプロセッサの命令セットシミュレータ（ISS：Instruction-Set Simulator）を用いてシミュレートすることが可能である。一般に、このようなシミュレータは、次の3つの要素を有する。

1. ターゲットプロセッサのオペレーションおよびレジスタのエミュレーション。
2. エミュレートされるプロセッサの状態を表示し、エミュレータのオペレーションを制御する、外部プログラムへのインタフェース、あるいはヒューマンインタフェース。
3. 試験対象システムに存在するデバイスを表す外部ハードウェアのエミュレーションへのインタフェース。

【0083】シミュレータの高度化と、外部デバイスとの相互作用とに依存して、結果は、試験対象システムの基本機能のみを確認することもあり、あるいは、すべてのプロセスのサイクル精度のシミュレーションを与えることもある。非サイクル精度のISSの構成を、本発明の第1の特徴として以下で説明し、本発明のさらに他の特徴による変形についてはその後説明する。

【0084】要素1は、ソフトウェアが、指定されたアーキテクチャによるプロセッサのレジスタをエミュレートすることを要求する。このようなレジスタには、汎用レジスタ、浮動小数点レジスタ、制御レジスタ、および、ユーザソフトウェアから直接にはアクセス可能でない「隠れた」レジスタが含まれるが、これらに限定されない。さらに、レジスタに対するオペレーションもまた、そのようなオペレーションによって引き起こされるステータスおよびエラー情報の生成を含めて、正しくエミュレートされなければならない。

【0085】要素2は、外部エージェントへのインタフェースを要求する。好ましくは、ISSは、非常に単純なコマンドラインインタフェースを有し、使用されるコマンドは、図5に示したテーブル1にリストされている。

【0086】要素3は、プロセッサに接続されたメモリおよびその他の外部デバイスのエミュレーションを要求

する。ISSは、2つの方法のうちの一方を用いてローカルRAMをエミュレートする。RAMの量が小さい場合、ホストマシンのローカルメモリが用いられる。そうでない場合、ディスクスペースを用いて、エミュレートされるRAMを保持する。これにより、実際のマシンより多くの物理メモリを有するマシンのエミュレーションが可能となる。また、これは、障害時のメモリの再検討可能なチェックポイントを提供する。予想されるように、外部ディスクを用いると、シミュレーションはかなり遅くなる。

【0087】ISSが外部ハードウェアに応答するためには、そのハードウェアのモデルにアクセスすることができなければならない。本発明の第1の特徴によれば、外部ハードウェアの動作を表すモジュールのセットが、ISS実行可能ファイルにリンクされる。外部デバイスは、外部グルーロジックを通じてCPUアドレス空間にマッピングされると仮定され、ある範囲の（非キャッシュ）メモリアドレスがデバイスへのアクセスを引き起こすようになる。このアクセス範囲は、ハードウェアコマンドによって定義される。

【0088】例えば、一般的なUARTインタフェースは、次の5つの機能を有することが可能である。

【0089】・void uart_reset(void)： このルーチンは、ハードウェアリセット信号の送信をシミュレートするために、ISS初期化中に呼び出される。

【0090】・uint uart_fetch(int ADDRESS, int LENGTH)： このルーチンは、プロセッサからデバイスへのリード（読出し）要求をシミュレートする。デバイスは、その入力アドレスライン上にADDRESSパラメータが提示されたかのように、LENGTHパラメータによって指定される適当な数のビットを返さなければならない。

【0091】・void uart_store(int ADDRESS, int LENGTH, uint VALUE)： このルーチンは、プロセッサからデバイスへのライト（書き込み）要求をエミュレートする。このルーチンは、ADDRESSパラメータがその入力アドレスラインに提示され、かつ、VALUEパラメータがその入力データラインに提示されたかのように、作用しなければならない。LENGTHパラメータは、有効な値のバイト数を指定する。

【0092】・void uart_tick(void)： このルーチンは、プロセッサクロックティックごとに呼び出され、データストリーミングやタイマオペレーションをシミュレートするために使用可能である。

【0093】・ハードウェアコンポーネントがプロセッサへの割込み信号を発生することを可能にする割込み発生ルーチン。

【0094】このエミュレーションは、タイミング情報を含まない。UARTは、イベントが提示されるのと同じクロックティック内に応答している。正確なタイミングが問題とはならない試験ソフトウェアの場合、この単

純なモデルで十分であった。

【0095】プロセッサは数百メガヘルツのクロックで動作することが可能であるが、このような速度は、命令およびデータがいずれも高速のオンチップキャッシュから来るときのみに実現される。遅いキャッシュや外部メモリにアクセスすると、CPUの見かけのパフォーマンスは低下することになる。さらに、多くのコデザイン問題の場合、外部ハードウェアは、オフチップRAMにのみ書き込みが可能である。このこともまた、ソフトウェア設計に対する制約となる。このようなデータにアクセスするにはデータキャッシュをバイパスしなければならない（結果としてそのデータはキャッシュにロードされるかもしれないが）からである。もちろん、メモリマップドI/Oロケーションへのアクセスもまた、非キャッシュアクセスを通じて行われなければならない。

【0096】しかし、このインタフェースは、コンポーネントに対する異なるアクセス時刻をエミュレートしない。すべてのメモリトランザクションは、単一サイクルで完了すると仮定される。

【0097】基本インタフェースは、必要に応じて個々の命令エミュレーションから呼び出される次の3つのルーチンを通じて提供される。

【0098】・void fetch(iblock *IN, uint ADDRESS1, int NBYTE, uint *ADDRESS2)：このルーチンは、（エミュレーション）アドレスADDRESS1からNBYTEバイトをロードし、それを（マシン）アドレスADDRESS2に格納（ストア）する。命令ブロックINは、リソーススコアボーディングのために用いられる。このルーチンは、キャッシュ可能領域ではデータキャッシュ探索を行い、それ以外ではエミュレートされたRAMやデバイスにアクセスする。

【0099】・void ifetch(uint ADDRESS1, int MAX, int IX)：このルーチンは、命令フェッチアルゴリズムをインプリメントする。これは、（エミュレーション）アドレスADDRESS1から始まるMAX個までのワードを、インデックスIXから始まる（4ワード）命令パイプにロードする。このルーチンは、まず、命令キャッシュを探索した後、データが見つからない場合に8ワードキャッシュラインを充填するためにメモリに進む。

【0100】・void store(block *IN, uint ADDRESS, int NBYTE, uint *VAL)：このルーチンは、（マシン）アドレスVALから（エミュレーション）アドレスADDRESSにNBYTEバイトを書き込む。また、このルーチンは、キャッシュ可能メモリのためにデータキャッシュを更新する。命令ブロックポインタINは、スコアボーディングのために用いられる。また、これらのルーチンは、キャッシュヒット数、キャッシュミス数、および非キャッシュメモリへのアクセス数についての統計を収集する。

【0101】タイミング測定が不要な場合、CPUの最

も単純（かつ最も高速）なシミュレーションは、命令を1個ずつフェッチおよびデコードし、次の命令に進む前に各命令の実行を完了することである。インテルI960プロセッサチップを例として用いると、ISSは、キャッシュ動作の十分なエミュレーションを提供する。これは、命令キャッシュおよびデータキャッシュの両方をサポートし、メモリコントローラレジスタ（mcon0~mcon15）を用いて、データをキャッシュするかどうかを決定する。また、このインタフェースは、メモリマップドハードウェアデバイスにアクセスするためのアドレス空間デコーディングも提供する。ルーチンは、ハードウェアルーチンのうちの1つ（例えば、uart_fetch）を呼び出すか、または、直接にRAMエミュレーションにアクセスする。

【0102】ISSは、軽量オペレーティングシステムの下で組込みアプリケーションとして動作する例示的な"hello world"プログラムを実行するために用いられた。この簡単なプログラムは、4個のプロセス、すなわち、UARTドライバ、コンソールプロセス、'hello'アプリケーション、およびアイドルプロセスを含む。また、このプログラムは、オペレーティングシステムのコアと、Cランタイムライブラリも含む。

【0103】ターゲットシステムは、開始エントリポイントから開始される。ターゲットシステムは、ブランク記憶領域をゼロにクリアし、制御されたプロセッサリセットを実行して、内部制御レジスタを設定する。初期化ルーチンが、4個のプロセスを作成し、UARTのためのハードウェア初期化ルーチンを呼び出す。このハードウェア初期化ルーチンは割込みハンドラをインストールする。

【0104】コンソールプロセスは、UARTプロセスへのメッセージチャネルを開き、UARTキーボードインタフェースからの入力を受け取ることができるようにメッセージのキューをセットアップする。

【0105】アプリケーションは、Cランタイムライブラリを用いて、通常のI/Oファイル（stdin、stdoutおよびstderr）をコンソールマルチプレクサプロセスに対して開いた後、printfルーチンを呼び出して、文字列をフォーマットしてメッセージをコンソールに送る。これは、コンソールへのコンテキストスイッチを引き起こし、文字列の前にプロセス名を付加してそれをUARTプロセスに転送する。

【0106】UARTは、ハードウェアによって生成されるTx-Available割込みに応答して、文字列を一時に1文字ずつハードウェアに送信する。すべての文字が送られた後、メッセージがコンソールプロセスを通じてアプリケーションに返され、その後、アプリケーションは終了する。

【0107】ISSをこのレベルで動作させると、ソフトウェアが正しく動作しているというある程度の信頼性

を示すことができる。例えば、アプリケーションとのU A R Tの動作を制御するソフトウェアは、いつCPUが文字を書くことができるかを示すためにチップによって供給されるステータスフラグを正しく解釈している。さらに、ルーチンは、正しいパラメータおよび連繋で、正しい順序で呼び出されている。また、検出可能なメモリ破損がなく、定義されたメモリ領域の外部へのアクセスもないと仮定することも妥当である。

【0108】しかし、このモードでシミュレータを動作させても、実行時間に関する実際の情報は得られない。あらゆる命令は1クロックティックしかかからないと仮定しているからである。命令タイミングモードを動作させ、パイプライントレースをプリントアウトすれば、命令サイクルがどこを進んでいるかが示される。

【0109】また、ISSを使用すると、回路内エミュレータを使用しても通常は入手できない情報にアクセスすることもできる。例えば、実際に使用されたレジスタキャッシュの最大深さを決定することや、キャッシュヒット統計とともにフレームこぼれ数をカウントすることも可能である。

【0110】また、それぞれの命令が実行された回数をカウントすることも可能である。このようなデータは、特に高集積SOCを製造するとき、プロセッサトレードオフを決定する際に非常に有益である。これは、例えば、より少数の分岐命令によって、ゲート数を少なくするためである。I960アーキテクチャは「予測分岐」命令を備えているが、コンパイラはそれらを使用しない。これらの命令を除去することも可能であり、あるいは、これらの機能を利用可能な別のコンパイラを検討することも可能である。

【0111】このレベルの詳細さでも、サイクル精度にはほど遠い。この場合、すべてのバストラッキングが（マルチワードオペレーションを含めて）同じサイクル内で完了するという1サイクルメモリモデルを仮定しているからである。このレベルでのタイミングを正確にモデル化するためには、システムデータバスとそのデバイスの精密なモデルが必要であり、これは、完全なコシミュレーション環境を生成することを必要とする。

【0112】次に、完全なサイクル精度の命令セットシミュレーションの構成について説明する。真のハードウェア/ソフトウェアコシミュレーションを可能にするためには、本発明の第1の特徴に要求されるISSの機能を少なくとも含む、選択されたCPUに対する命令セットシミュレータが要求される。

【0113】しかし、この機能は、特定のCPUを選択する際に次の3つの主要な要素を無視している。

【0114】1. 例えば乗算や除算のように、多くの命令は、単一サイクルで実行を完了しない。

【0115】2. CPUは、リソース使用衝突がないと仮定して、バスオペレーションがレジスタ間オペレーシ

ョンと同時に実行可能なように、複数の専用ユニットにわたる内部並列処理をインプリメントしていることがある。

【0116】3. 内部キャッシュメモリ、外部RAMメモリおよびデバイスへのアクセスは、互いに非常に異なる応答時間を有することがある。これによる通常の効果は単に実行が遅延されるだけであるが、ある重大な場合には（例えば、到来する割込みの相対的タイミングがあるタスクの処理順序を決定する場合）、全体のパフォーマンスにおいて多大な役割を演ずることがある。

【0117】次に、電子回路とその電子回路を制御する制御プログラムとを有するシステムの検証のための、ターゲットプロセッサのサイクル精度の命令セットシミュレーションを導出する方法について詳細に説明する。

【0118】インテルI960プロセッサを例として用いると、ISSは、命令キャッシュに加えて、次の機能をインプリメントすることによって、I960命令スケジューラをモデル化する。

- ・並列デコーディングを有する4ワード命令パイプ。
- ・5個のI960命令バスのエミュレーション。
- ・結果が利用可能になる前には使用されないことを保証するためのレジスタスコアボーディング。
- ・分岐や呼出し（コール）のようなパイプライン切断命令の実装。
- ・完了するのに単一クロック時間より長くなる命令に対するパイプラインストール。

【0119】図6を参照すると、命令パイプラインは、各プロセッサクロックティックの最初に検査される。パイプが完全に満たされていない場合、満たすためにifetchルーチンが呼び出される。パイプを満たすのに必要な命令が命令キャッシュに見つからない場合、ifetch擬似命令がスケジューリングされる。この命令は、CPUのバス制御ユニット（BCU: Bus Control Unit）およびメモリバスにおいて実行される（そのため、これらのユニットがビジーである場合には遅延されることがある）。結果的に、この命令が実行されると、8ワードが命令キャッシュラインにロードされる。これらのワードのうちの4個までが、fill_lpipeルーチンによって、命令パイプラインを満たすために使用される。使用されるワード数は、パイプの状態と、命令キャッシュバウンダリに関する現在の命令ポインタのアラインメントとに依存する。

【0120】命令パイプラインにデータがある場合、その値が命令へとデコードされる。一部のI960命令は複数ワードを占めるため、不完全な命令がパイプに存在する可能性があり、その場合（図6の第2のcallx命令の場合のように）、それはデコードすることができない。命令は一度だけでコードされ、デコードされた形がデータ構造体に格納される。また、デコーディングには、命令をスケジューリングするために必要な内部処理

ユニット、パイプラインおよびレジスタリソースを識別することも関連する。

【0121】CPUが使用中のリソースは、ティックごとに、スコアボード構造体によって表される。命令は、パイプラインから順に取り出され、1つずつスケジューリングされる。スケジューリングは、命令がこのティックの間にすでにコミットされたリソースを要求するとき、もしくは、命令がパイプライン切断を生じるとき、または、パイプライン内にもはやデコードされた命令がないときのいずれかに、停止する。

【0122】スケジューラがパイプライン内のスロットを消費した場合、残りのデータが繰り上がり、命令ポイントが更新される。

【0123】スケジューリングが完了すると、現在のティックにおいてユニットに入る命令が実行される。完了するのに単一ティックより長くかかる命令は、パイプラインがストールすると、そのリソースを後続のティックに伝搬させる。

【0124】図6を参照すると、AGU、MEMバス、ソースレジスタとしてのフレームポインタ、および、デスティネーションレジスタとしてのスタックポインタをロックして、lda命令が現在のティックにおいてスケジューリングされることが可能である。これは、次のcallx命令がスケジューリングされないようにするため、次のcallx命令はパイプにとどまる。lda命令は、実行するのに複数サイクルかかるため、AGUは次のティックでストールし、この命令からのリソースは、この命令がAGUを占有する限り、スコアボードにコピーされる。ldaは、完了すると、そのリソースを解放し、callx命令が開始することができる。次のcallx（これはここで完全にデコードされることが可能となる）は、次の2つの理由で、スケジューリングされない。第1に、このcallxでのリソースクラッシュのためであるが、それだけでなく第2に、callx命令はパイプライン切断（および新しい命令ポイントへの転送）を引き起こすからである。

【0125】図6を参照すると、lda命令のエミュレーションは特に複雑ではない。命令デコーディングルーチンは、デスティネーションレジスタおよびメモリオペランドをすでに評価しているため、ルーチンは、レジスタを保持している（ISSメモリ内の）ロケーションを単に更新するだけである。その結果は、検査のために外部観察者に対して表示することが可能であり、スコアボードは、この命令に対する全部で4ティックについてマークされる。

【0126】エミュレーションにおいては、デスティネーションレジスタが命令の期間中にアクセス不能とマークされている限り、現実のCPUの場合のように「最終」ティックまで結果の代入を遅延させる必要はない。この便法は、ハードウェアがコシミュレートされているときにはもはや成り立たない。

【0127】図7～図9を参照して、サイクル精度のターゲットプロセッサのモデル化のプロセスについて説明する。S1105で、ターゲットプロセッサがオンボードキャッシュメモリを有するかどうかを判定する。ターゲットプロセッサがオンボードキャッシュメモリを有する場合、プロセスフローはS1115に進み、オンボードキャッシュアクセスを制御するアルゴリズムがモデル化される。ターゲットプロセッサがオンボードキャッシュメモリを有しない場合、プロセスフローはS1120に進む。

【0128】S1120で、ターゲットプロセッサが命令パイプラインアーキテクチャを有するかどうかを判定する。ターゲットプロセッサが命令パイプラインアーキテクチャを有する場合、S1130で、命令パイプライン充填プロセスがモデル化される。そうでない場合、プロセスフローはS1135に進む。

【0129】S1135で、命令およびリソースの表現を作成する。ターゲットプロセッサのアーキテクチャに依存して、各命令の効果、その命令が要求しロックするターゲットプロセッサ内のリソース、および、その命令の継続時間または終了条件を表現するデータ構造体が定義され初期化される。

【0130】図8を参照すると、S1140で、上記で定義されたリソース定義と、現在のマシン状態における命令のアクションとに対するインタプリタを作成することによって、命令がシミュレートされる。

【0131】S1145で、ターゲットプロセッサがマルチバスアーキテクチャを有するかどうかを判定する。ターゲットプロセッサがマルチバスアーキテクチャを有しない場合、プロセスフローはS1165に進む。これに対して、ターゲットプロセッサがマルチバスアーキテクチャを有する場合、S1155で、現在のマシン状態において、他の命令が並列に実行されるべきかどうかを判定するアルゴリズムが生成される。他の命令が並列に実行されるべきである場合、プロセスフローはS1140に戻り、並列命令が実行される。そうでない場合、プロセスフローはS1165に進む。

【0132】図8および図9を参照すると、S1165で、シミュレートされる命令がマルチサイクル命令であるかどうかを判定する。シミュレートされる命令がマルチサイクル命令である場合、S1170で、上記のようにスコアボードが用意される。次に、S1175で、上記のようにリソースが伝搬され、プロセスフローはS1180に進む。

【0133】S1180で、非決定性タイミングが存在するかどうかを判定する。非決定性タイミングが存在する場合、終了の評価を行う。S1190で、現在のマシン状態における非決定性タイミングオペレーションに対する終了条件を判定するアルゴリズムを生成する。ターゲットプロセッサのアーキテクチャに従って、これは、

エミュレートされるマシンレジスタ内の値に基づく計算を含むこともあり、また、外部信号の状態を調べるコードを生成することを含むこともある。S1192で、終了条件がチェックされ、命令が終了していない場合、プロセスフローはS1175に進む。命令が終了している場合、前に伝搬されたリソースがクリアされ、命令のシミュレーションは完了する。

【0134】図10～図13を参照して、電子回路と、その電子回路を制御する制御プログラムとを含むシステムのコバリデーションの実行について説明する。S1310で、刺激が試験対象システムに入力される。S1315で、CPUが「動作可能」であるかどうかを判定する。動作可能である場合、プロセスフローはS1340に進む。動作可能でない場合、S1320で、ターゲットプロセッサはストールされ、S1325で、エミュレートされているキャッシュメモリに命令がロードされる。S1330で、キャッシュメモリがロードされているかを判定する。メモリが十分にロードされている場合、S1335で、ターゲットプロセッサがアンストール（ストール解除）される。

【0135】S1340で、命令パイプラインが、ターゲットプロセッサの内部データフローモデルから導出される信号のシーケンスを用いて充填される。図11を参照すると、S1345で、ターゲットプロセッサの内部データバスのアクションをエミュレートするために、所定数のクロックサイクルに基づく遅延が実行される。この所定数は、内部データバスの幅に基づき、1～数クロックサイクルの範囲とすることが可能である。

【0136】次に、S1350で、実行に利用可能な命令を解釈するために、ターゲットプロセッサの内部デコードサイクルが実行される。この命令デコードサイクルは、上記のように実行される。

【0137】ターゲットプロセッサの内部デコードサイクルが実行された後、S1355で、スケジューリングされた命令が上記のようにパイプライン切断をインプリメントしているかどうかを判定する。パイプライン切断が要求される場合、S1365で、命令スケジューラがストールされる。パイプライン切断が要求されない場合、プロセス制御フローはS1370に進む。

【0138】次に、S1320で、スケジューリングされた命令が、適当な命令パイプラインあるいはハードウェアコンポーネントに転送される。適当な命令パイプラインへの命令の転送後、S1375で、各命令のサイクル実行時間が計算される。

【0139】図12を参照すると、S1380で、非決定性タイミングが存在するかどうかを判定する。非決定性タイミングが存在する場合、S1390で、S1190で生成された終了条件を判定するアルゴリズムが、各クロックサイクルの終端で実行される。

【0140】S1400で、エミュレーションは、計算

された命令時間に対するリソース割当てを伝搬させる。S1405で、計算された実行サイクルの終端で利用可能な結果が出力される。この結果は、エミュレートされたレジスタやフラグの値を更新することや、外部信号の値を変化させることを含むことがある。

【0141】次に、S1410で、次の命令サイクルで割込みハンドラがスケジューリングされるべきかどうかを判定する。スケジューリングされるべきである場合、S1420で、次の命令サイクルで実行するために割込みハンドラがスケジューリングされる。

【0142】図13を参照すると、S1425で、刺激からの結果が試験対象システムから出力される。この出力は、外部観察者に対して示されるシミュレーションの進行の視覚的表示であることも可能であり、あるいは、後の検討や後処理のためのそのような変化のログ（例えば、図20によって例示されるようなトレース出力）の生成であることも可能である。S1430で、試験対象システムに入力されるべき刺激がまだあるかどうかを判定し、もうない場合、プロセスは終了する。まだある場合、プロセス制御はS1310に進む。

【0143】次に、試験対象システムへの刺激の入力により、試験対象システムに結果を出力させることについて詳細に説明する。

【0144】一般に、いくつかの異なるプロセスが、試験対象システムの異なる部分をシミュレートする。これらのプロセスは試験対象システムに固有であり、ケースバイケースで修正されなければならない。図14を参照すると、設計された試験サイクルを継続的に繰り返して、コシミュレートされるハードウェアおよびソフトウェアを十分に作用させるために、例として、グローバルブコントロール23が用いられる。試験刺激プロセス24は、「外界」から試験対象システムに刺激を提供する。試験刺激プロセス24のもう1つの機能は、誤った入力がどのように処理されるかを見るために、誤入力を試験対象システムに入力することである。例えば、試験刺激プロセス24は、データ転送トランザクションの応答側として作用し、試験対象システムの再試行メカニズムを試験するために、プロトコルエラーで応答することが可能である。バスアービトレーションプロセス22は、試験対象システムと試験刺激プロセスの間のバスアービトレーションを提供する。試験対象回路21は、単一のプロセスであることも可能であり、あるいは代替例では、回路設計の複雑さに依存して、数個のプロセスであることも可能である。

【0145】さらに、信号表示プロセスにより、ステップ実行を行い、グローバル状態に対する個々のステートメントの効果をウォッチすることが可能である。

【0146】次に、電子回路のハードウェアモデルを命令セットシミュレーションに結合して、軽量コンピュータオペレーティングシステム環境で実行される試験対象

システムを作成することについて、さらに詳細に説明する。

【0147】第1段階として、ハードウェアの動作記述を調べ、各ハードウェアプロセスが個別のソフトウェアプロセスによって表現されるプロセス構造が構成される。各プロセス内では、動作記述が、軽量オペレーティングシステムのプログラミング言語に翻訳される。命令セットシミュレータがこの言語で同様にコーディングされる。シミュレータに必要なファブリックプロセスおよび刺激生成プログラムも同様である。第2段階として、プログラミング言語ソースファイルは、軽量オペレーティングシステムが動作するプロセッサに対するオブジェクトコードにコンパイルされる。第3段階として、これらのファイルが、軽量オペレーティングシステムに対するオブジェクトコードとリンクされ、完全な実行環境が形成される。第4段階として、シミュレータが実行されるシステムに、実行可能ファイルがロードされる。第5段階として、シミュレータが、軽量オペレーティングシステムの制御下で実行され、出力が観察される。

【0148】次に、図14を参照して、電子回路と、その電子回路をターゲットとする制御プログラムとのコバリデーションのためのコンピュータシステムについて、簡単に説明する。電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされる。また、このコンピュータシステムは、上記のルーチンを使用する。S0900の設計ステップの終了後、コンピュータシステムは、ファブリックプロセス10を有する。ファブリックプロセス10は、複数のグローバル信号の状態を管理する信号メンテナサブシステム12と、ソフトウェアモデルおよび命令セットシミュレータによって使用されるクロック信号を生成するクロックジェネレータサブシステム13と、ソフトウェアモデルおよび命令セットシミュレータからのイベントをウェイトするプロセスキューを管理するキューメンテナサブシステム14と、所定のタイミング間隔で少なくとも1つのプロセスを実行するスケジューラサブシステム16と、少なくとも信号メンテナサブシステム12、クロック信号ジェネレータサブシステム13、キューメンテナサブシステム14およびスケジューラサブシステムの実行を制御する中央制御プロセス11とを有する。

【0149】マイクロカーネルサポートプロセス以外に、コンピュータシステムは、ファブリックプロセス10、命令セットシミュレーションプロセス、ならびに、ハードウェアコンポーネントおよび刺激システムを表すプロセスを有し、これらはすべて互に関連して動作する。コシミュレーション環境の中央制御プロセス11は、次の5個の主要なサブシステムの全体の機能および相互作用を管理する。

(a) すべてのグローバル信号の状態を管理する信号メ

ンテナ12。

(b) クロック信号を生成するクロックジェネレータ13。

(c) イベントをウェイトするプロセスのキューを管理するキューメンテナ14。

(d) 各ナノ秒ティックごとにそれぞれのアクティブプロセスを実行するスケジューラ16。

(e) ユーザに対して信号変化を示すディスプレイを駆動するディスプレイジェネレータ15。

10 試験刺激の例としては、外部インタフェース上でのデータ利用可能性を示す信号がある。また、刺激は、エラー条件、例えば、ECCメモリによって生成されるパリティ検査エラーをシミュレートすることも可能である。

【0150】図15を参照すると、シミュレーション環境の全体は、軽量オペレーティングシステム30内の複数のプロセスとして動作する。個々のプロセスには、ファブリックプロセス31、命令セットシミュレーションプロセス32およびVHDLプロセス33、34が含まれる。VHDLプロセス33、34は、モデル化されているハードウェアのインスタンス化されたハードウェアプロセスと一対一に対応する。すべての信号相互作用35は、オペレーティングシステム30によってスケジューリングされたメッセージとして、ファブリックプロセス31を通る。

20 【0151】このようなソフトウェアベースのシミュレーションシステム内では、シングルフロッピッドシステムでは真の並行性はないため、見かけ上の並列処理しか達成することができない。これは、信号に対する「同時」変化と時間の概念の問題を引き起こす。考慮すべき次の3つの問題がある。

30 【0152】・問題1： 2つのプロセスが動作可能であり、それらの実行の期間中に、ある信号の値を変化させる場合、その結果は、プロセスが実行される順序に依存することがある。例えば、信号"test"が実行の最初に値'1'を有する場合、

プロセス1： if (test = '1') then test <= '0'; end if;

プロセス2： test <= '1';

40 【0153】プロセス1がプロセス2の前に実行される場合、(素朴な)結果は、実行の終了時に"test"が'1'にセットされるというものである。プロセス2がプロセス1の前に実行される場合、結果は、実行の終了時に"test"が'0'にセットされるというものである。

【0154】・問題2： プロセスが変化し、同じ実行期間に、ある信号を読み出す場合、結果は非決定性となる。例えば、次のコードフラグメントを考える。

プロセス1： test <= '1';

if (test = '1') then output <= '1'; end if;

50 現実のシステムでは、これはoutputを'1'にセットすることもしないこともある。

【0155】・問題3： プロセスは、既知の周波数で動作するクロック信号に対する絶対時間の値を必要とする。さらに、絶対時間は、より複雑なデューティサイクルとともに、次のような構文に必要である。

クロックプロセス: `clk <= !clk after 5 ns;`

【0156】従来のVHDLシミュレータは通常、最初の2つの問題を、「デルタ」、すなわち、サブティック同期ポイントの概念により処理する。サブティック同期ポイントを使用すると、すべての信号値は、デルタの最初に固定される。これらの値が信号値に対する要求に回答して出され、デルタの最後にのみ値が更新される。これは、シミュレーションを決定性にし、プロセスのシミュレーションの順序とは独立にするという利点がある。これは、VHDLプログラマに対して、次のデルタへのアラインメントを強制するように余分のウェイト(wait)ステートメントをコードに挿入することを要求する。上記の問題2の場合、この非決定性問題を解決するために、デルタバウンダリを強制するステートメントが挿入される。

プロセス 1 : test <= '1';

```
wait for 0 ns;
```

```
if (test = '1') then output <= '1'; end if;
```

プロセス1は一貫してoutputを'1'にセットすることになる。しかし、合成された結果が期待通りに動作するという保証はない。

【0157】ファブリックプロセスの現在の実装では、瞬時値が出される（そのため、この例は、ウェイトが暗黙のうちに挿入されているかのように動作する）が、結果は、プロセスの実行の順序に依存する。当業者には明らかなように、システムは、1つの信号に対する単一の「デルタ」内の複数の変化を検出して、これをシミュレーション出力に報告するか、あるいは、他の動作をまねるように実装を変更するように、修正することも可能である。

【0158】シミュレータは、絶対時間の値を生成する。ファブリックプロセスのクロックサブシステムは、現在、1ナノ秒の基本タイムスライスを使用しているが、これは、現在のクロック信号の必要な精度を追跡するのに十分である。これは、数GHzのシステムがさらに広まるに依じて、変更される必要があるかもしれない。クロック信号のサポートは、共通のオペレーションのためのインタフェースを単純化するために、中央プロセスに組み込まれた。

【0159】ほとんどの動作VHDLは、2つのモードのうち的一方で動作するトリガで書かれる。非同期モードでは、信号変化は、その信号に反応するプロセスを、次のナノ秒で動作可能にした。同期（クロック）モードでは、信号は、クロック信号の端（立上りあるいは立下り）でのみ調べられた。これらのモードは、図2のS1000における動作翻訳中に決定される。プロセスの動

作可能性は、前のティックに生じた信号変化と、そのときに起こっているクロック遷移とに基づいて、各ナノ秒ティックの最初に決定された。そのナノ秒ティック中に動作可能となったプロセスは次のティックまで開始されず、「デルタ」メカニズムに類似の環境提供した。

【0160】このインタフェース方法の顕著な特徴は、イベントをウェイトしているプロセスが、マイクロカーネルコア内で真のウェイト状態にあることである。すなわち、それらのプロセスは、各クロックティックでチェックされず、システムの残りの部分におけるオーバーヘッドを生じない。ファブリックプロセスは、条件のすべてのチェックを処理する。従来のオペレーティングシステムのオーバーヘッドを除去し、軽量オペレーティングシステムの直下でシミュレーションを実行することによって、本発明は、より高速に動作するのみならず、エミュレートされる状態の表現は、オペレーティングシステム自体のスケジューリングコアを通じて直接にインプリメントされる。

【0161】個々のプロセスが翻訳されたコードを実行するため、トレース出力はコンソールに表示することができ、システム全体のECHOVHDL変数が0でない場合、シミュレーション環境は、ナノ秒ティックのタイムスタンプを前に付した文字列を印字することによって信号値の変化が観察者に見えるようにする。コシミュレーション環境内の信号表示サブシステムと組み合わせて、ステップ実行を行い、グローバル状態に対する個々のVHDLステートメントの効果をウォッチすることが可能である。

【０１６２】図１６に、プロセッサ４０、Ｉ／Ｏデバイス４３およびビデオディスプレイ端末４１を有するコンピュータシステムの実施例を示す。Ｉ／Ｏデバイス４３は、キーボードおよびマウスを含むが、これらに限定されない。タッチパッドのような他のデバイスも使用可能である。さらに、コンピュータシステムは、このコンピュータシステムが本発明のステップを実行することを可能にするように適応したソフトウェア命令を含むメモリ４２（図示していないが、プロセッサ４０に組み込まれている）を有する。

【0163】また、コンピュータシステムは、データリンク44によってプロセッサ40に接続されたサーバ45も含むことが可能である。データリンク44は、従来のデータリンク（例えば、イーサネット、ツイストペア、FTP、HTTPなど）である。サーバ45は、このサーバに接続されたプログラムライブラリ46へのアクセスを提供する。また、プログラムライブラリ46は、コンピュータシステムが本発明のステップを実行することを可能にするように適応したソフトウェア命令を提供することも可能である。上記のように、プログラムライブラリ46は、当業者に周知の任意のさまざまな媒体（例えば、フロッピー（登録商標）ディスク、ハード

ディスク、光ディスク、カートリッジ、テープ、CD-ROM、書き込み可能CDなど)上に実現可能である。図16に示したコンピュータシステムでは、メモリ42上のソフトウェア命令により、プロセッサ40は、データリンク44を通じてサーバ45にアクセスすることによって、プログラムライブラリ46にアクセスすることができる。図16に示したコンピュータシステムは、いかなる意味でも限定的であることは意図しておらず、本発明を実施する多数のさまざまなコンピュータシステムを組み合わせることが可能である。

【0164】図17に、本発明によってモデル化される例示的なシステムのブロックレベル設計を示す。システムは、I960プロセッサ35、グルーロジック36およびUART37を有する。この例示的なシステムは、本発明の適用範囲を制限するものではなく、以下の説明に役立てるためのものである。このサンプルシステムは、外部データバスがメモリサブシステム38およびアドレスデコーディンググルーロジック36の両方に接続されたI960命令セットシミュレータを有する。矢印は、信号の一方方向性または双方向性を示す。グルーロジック36は、メモリおよびUARTチップ37に対するchip-select信号をドライブするとともに、UART37に対してさまざまな信号提示要求を提供する。この設計は、プロセッサクロックおよびバスクロックという2つのクロックにより、意図的に複数のタイミングドメインに分割された。

【0165】コシミュレーション環境を実証するため、I960命令セットシミュレータは、NEC VR4300プロセッサのユーザガイド(User Guide)からの情報を用いて、MIPSアーキテクチャに基づくデータバスにリンクされた。バス上には外部バスマスタリングデバイスがなかったため、システムは、フルMIPSバスのサブセットを使用した。選択された信号のセットは、さまざまなバースト能力とともに、さまざまな応答時間で、デバイスをインプリメントするのに十分であった。

【0166】バスは同期モードで動作し、アクティビティはbusclk信号の立上り端でスケジューリングされる。信号およびプロトコルは以下に記載するとおりである。次のバス信号が用いられる。

・SysAD [31..0] 多重化システムアドレスおよびデータバス。

・SysCmd [4..0] データおよびトランザクションタイプを識別するコマンドバス。これは、リードモードに8ワードのバーストを追加するが、エラー指示コマンドおよび外部非応答モードを使用しない。

・EOK 外部デバイスがコマンドを受け入れることができるときにローにセットされる。

・EValid 外部デバイスがバス上に有効なデータを置いたときにローにセットされる。

・PMaster プロセッサが(外部デバイスがリード応答

を発行することができるように)バス所有権を解放したときにハイにセットされる。外部バスマスタリングデバイスのないこの簡単なモデルでは、フルMIPSバスからの他のプロトコル信号は使用されなかった。

【0167】プロセッサの視点から、図18を参照すると、バスは次の順序でドライブされる。

1. プロセッサは、EOKがローになる(これは、バスを使用中の外部デバイスがないことを示す)のをウェイトする。

2. プロセッサは、SysAD上にアドレスを、また、SysCmd上に適当なWriteコマンドをドライブする。また、プロセッサは、PValidをローにセットする(これは、バス上に有効なデータがあることを示す)。

3. プロセッサは、EOKがハイになる(これは、外部デバイスがコマンドを受け入れたことを示す)のをウェイトする。

4. プロセッサは、SysADバスに、busclkサイクルごとに1個ずつデータワードを順次ドライブし、最後(または唯一)のワードを'no-more'フラグでマークする。

5. 最後のデータサイクルの終わりに、プロセッサは、PValidをハイにセットし(これは、プロセッサがもはやバスをドライブしていないことを示す)、SysADおよびSysCmdをトラリステートにする。

【0168】図19を参照すると、プロセッサリードは次のようになる。

1. プロセッサは、EOKがローになる(これは、バスを使用中の外部デバイスがないことを示す)のをウェイトする。

2. プロセッサは、SysAD上にアドレスを、また、SysCmd上に適当なReadコマンドをドライブする。また、プロセッサは、PValidをローにセットする(これは、バス上に有効なデータがあることを示す)。

3. プロセッサは、EOKがハイになる(これは、外部デバイスがコマンドを受け入れたことを示す)のをウェイトする。

4. プロセッサは、PValidをハイにセットし(これは、プロセッサがもはやバスをドライブしていないことを示す)、PMasterをハイにセットしてバスの制御を解放する。

5. プロセッサは、EValidがローにセットされる(これは、外部デバイスがバス上に有効なデータを置いたことを示す)のをウェイトする。次に、プロセッサは、SysADバスから、busclkサイクルごとに1個ずつデータワードを順次読み出す。

6. プロセッサは、EValidがハイにセットされるのをウェイトしてから、PMasterをローにセットし、バスの制御を再び獲得する。

【0169】UARTは、次のような非常に単純な信号モデルを使用した。

uartD [7..0] は、チップとの間での単一データ(非

バースト) アクセスをサポートするバイト幅双方向データバスである。

・uartAは、2つの内部レジスタのうち的一方を選択するために用いられる1ビット「アドレス」バスである。アドレス'0'は制御レジスタであり、アドレス'1'はデータレジスタである。

・uartRは、プロセッサがリード要求を発行した場合にローにセットされる。

・uartWは、プロセッサがライト要求を発行した場合にローにセットされる。

・uartSは、UARTを選択するためにローにセットされる。アドレスおよびデータバスの値は、チップが選択されている間中有効でなければならない。

・uartIは、UARTからプロセッサに割込みを通知するためにハイにセットされる。

【0170】busclkは、チップに対する同期クロック信号であり、すべての遷移は、クロックの立上り端でサンプリングされる。

【0171】UARTは、アドレスおよびデータの値が同時に提示されることを要求するため、グルーロジックは、多重化されたSysADバスからの翻訳を処理しなければならない。

【0172】

【表1】

テーブル2

ライト	0x38レジスタ0	リセット割込み
ライト	0x02レジスタ1	割込み送信イネーブル
リード	0x04レジスタ0	バッファエンプティ送信

【0173】制御レジスタは、まずインデックス番号を書き込んだ後に8ビットのデータを書き込むという2段階アクセスによって、内部UARTレジスタにアクセスするために用いられる。制御レジスタ0は、制御アドレスに制御オペレーションを書き込むことによって直接にアクセスされることが可能である(制御オペレーションは、間接レジスタ番号の範囲外のバイナリ値を有する)。この最小モデルでは、上記テーブル2にリストした制御オペレーションのみがインプリメントされた。これらは、割込み駆動およびポーリングモードの出力(入

力はない)を提供するのに十分であった。

【0174】システムメモリは、busclkの速度でSysADラインを通じてのバックツージャンプモード転送をサポートすると仮定された。メモリコントローラには、プロセッサバス上のSysAD、SysCmd、PMasterおよびEValid信号へのアクセスが許可され、busclkでクロックされた。メモリコントローラは、グルーロジックへのさらに2つの次の信号を使用した。

・memSはメモリchip-selectであり、これは、メモリが、バス上のコマンドおよびアドレスデータのターゲットであったときにローにセットされる。

・memEOKは、メモリコントローラがコマンドを受け入れることができるときに、メモリコントローラによってローにセットされる。

【0175】「現実の」コントローラの内部の詳細(例えば、RASおよびCAS生成)はモデル化されず、すべてのメモリアクセスは単一の(バス)クロックサイクルで応答された。

【0176】UARTおよびメモリの実装はいずれも、上記のアーキテクチャ仕様によるそれらのオペレーションを制御するために、VHDL信号ライブラリを用いて直接にbehavioral Cで書かれた。さらに、UART実装は、ISSログファイルに出力されるように命令された文字列を格納した。このことは、シミュレータは、シミュレートされたシステムの出力の視覚的記録を生成したことを意味する。グルーロジックは、アドレスでコーディングおよび信号翻訳のための状態マシンを実装するVHDLで、単一のエンティティとして書かれた。VHDLは、S1000の方法に従ってCに翻訳された。

【0177】次に、上記の命令セットシミュレータをコシミュレーション環境に統合することについて説明する。ISSをファブリック環境に統合するためには、ISSにいくつかの変更が必要とされた。いったんI/Oオペレーションが予測可能な継続時間のものではなくなると、その変更のほとんどは、タイミングの変更に関するものであった。

【0178】ファブリックインタフェース、プロセスの同期クロックおよび信号のセットを初期化するために、次のコードがプロセス初期化シーケンスに追加された。

【0179】

```
FabInit();
/* pclock周期を指定する */
fabric_set_clock("pclk", 0, 5);
/* プロセッサ信号の初期状態 */
Signal(PMaster, 0);
Signal(PValid, 1);
/* ドライブされるまでデータおよびコマンドバスは浮遊 */
SignalZ(SysAD);
```

41

SignalZ(SysCmd);

【0180】単一のプロセッサクロックティックをシミュレートするルーチンは、各ティックを実行する前に、WaitClkR();

への呼出しを追加することによって、中央ファブリックプロセスからのpclk上の立上り端変化をウェイトするように変更された。これはまた、プロセッサを他のハードウェアコンポーネントと同期させる効果を有する。注意すべき点であるが、このオペレーションのシーケンスは、選択されたデータバスモデルのアーキテクチャによって決定されているため、モデル化されているシステムに固有である。このような統合ステップは、コシミュレートされるそれぞれの設計ごとに実行されることになる。

【0181】入出力（フェッチおよびストア）オペレーションの基本的なストラテジは同じままであった。メモリ領域デスク립タに依存して、データキャッシュ（または命令キャッシュ）をまず参照した後、データがキャッシュにない場合には、外部デバイスのうちの1つがアクセスされる。主な変更は、すべてのアドレスデコーディングがグルーロジックで行われ、「ハードウェア」記述およびdevice_storeルーチンが使用されなかったことである。

【0182】インタフェース自体は、上記の説明に従った。例えば、新たなオペレーションは次のように開始された（変数thisはI/O要求データ構造体を指すと仮定する）。

```

【0183】
    if (GetSignal(R0K) == 0)
    {
        Signal(Pvalid, 0);
        Signal(SysAD, this->addr);
        Signal(SysCmd, this->op);
        this->memstate = 1;
    }
    sbmarksb(this->inst, 2);

```

すなわち、EOKがローにセットされるのをウェイトした後、バス上に信号をドライブする。なお、コマンドが開始されたかどうかにかかわらず、ユニットは次のティックの間ストールされ、リソースは利用可能でないとマークされなければならない。また、このルーチンはそのすべての信号に対してポーリングを行う。ウェイトポイントが'step'ルーチンにあるからである。

【0184】外部バスマスタリングデバイス、あるいは、エラー処理がない場合、外部インタフェースは、状態が10個より少ない小規模な状態マシンで捕捉された。また、このマシンは、命令フェッチサイクルも処理した。

【0185】このようなシステムに割込み信号を統合す

る際には特別の注意をしなければならない。シングルプロセス実装では、シミュレートされるプロセッサの割込み処理中の制御レジスタを直接修正することが可能である。コシミュレーションシステムでは、これは、UARTコードにおける

```

Signal(uart1, 1);
への呼出しと、ISSプロセスにおける対応するテスト
    if (GetSignal(uart1) == 1)
    {
        IGEN(7);
    }

```

によって置き換えなければならない。このアプローチは、バス制御ユニット(BusControl Unit)がビジーでありプロセッサが割込みを直ちに処理することができないとき、または、割込みが実行可能になる前に命令フェッチを要求するときに問題を引き起こした。特別の対処がなければ、プロセッサの優先度レジスタが割込み優先度に更新されるまで、割込みは各クロックティックにおいて処理されることになる。

【0186】これは、割込み処理が開始されるとセットされるフラグにより処理された。このフラグは、バス制御ユニットにおけるアクティビティ（割込みデイスパッチャによってスケジューリングされるアクティビティを含む）がある限り、すべてのパイプラインスケジューリングをバイパスした。このフラグをセットすることは、割込み処理中のレジスタを検査するオペレーションが再びスケジューリングされる前にすべてのI/Oオペレーションが完了してプロセッサ優先度が更新されることを保証した。

【0187】プロセッサクロックは、100MHzで動作するようにセットされた。プロセッサ内では、命令キャッシュと命令パイプの間の内部バスは、フルスピードで動作するフル128ビット幅と仮定された。したがって、命令パイプはキャッシュからレイテンシ0で充填される。データキャッシュは、レベル1、すなわち、フルプロセッサスピードにあると仮定された。メモリアクセスはbusclkで動作したため、アクセスあたり1個の（プロセッサ）ウェイト状態を必要とした。

【0188】UARTは、9600ボーの外部シリアルラインに接続されていると仮定された。したがって、文字は1ミリ秒ごとに送出された。UART内部にはバッファリングがないため、'Tx Available'信号は、ライトオペレーションごとに1ミリ秒だけ遅延された。

【0189】このシステムは、前述のコード例、すなわち"Hello World"プログラムの実行を繰り返すために使用された。図20に示す出力は、MIPS-IIIアーキテクチャCPUを含む開発ボード上で実行された、動作中のシステムを実証している。このアーキテクチャは、シミュレートされているCPU（インテルI96

0) のアーキテクチャとは異なり、より能力の高いプロセッサがどのようにシミュレーションのために用いられるかを実証している。

【0190】ターゲットCPU自体の処理能力が低い場合、ターゲットCPUは、シミュレーション環境をサポートしておらず、適当な時間で結果を出さない可能性がある。軽量オペレーティングシステムのためのデバイスドライバを容易に書くことができるため、迅速なプロトタイピングとシミュレーション環境が提供可能である。シミュレータをシングルCPUに制約する理由はない。

【0191】軽量オペレーティングシステムコア自体を除いて、シミュレーションシステムは、特定のマシンアーキテクチャとは独立である。その非常に低いオーバーヘッドおよびメモリ要求により、軽量オペレーティングシステムは、ページングやスワッピングによらずに、利用可能なRAMをシミュレーションのために最大限に利用させることができる。また、このようなシステムは一般に、最小限の外部周辺装置のセットしか必要としないため、従来のワークステーションのためのシミュレーションエンジンあるいはアクセラレータとして高性能CPUの利用することも考えられる。

【0192】図16を参照すると、本発明のもう1つの特徴によれば、目的とするターゲットCPUは、通信媒体51を通じてシミュレーションシステム40に接続された開発ボード50上に位置する。通信媒体51は、システム40内に共存するデータベースであることも可能である。ターゲットアプリケーションは、開発ボード50のCPU上で直接実行され、命令セットシミュレータの必要性が回避される。このアプローチは、サイクルレベルの制度を維持するためにCPU命令セットからのサポートを必要とし、任意の与えられたCPUに対するその実現は、すでに説明した原理から導出することが可能である。

【0193】本発明のさまざまな特徴についての以上の記載は、例示および説明の目的で提示したものである。これは、網羅的であることや、開示したとおりの形に本発明を限定することは意図しておらず、さまざまな修正および変更が、上記の説明に照らして可能であり、また、本発明の実施により得られる。当業者が、考えている個々の用途に合わせて、さまざまな実施例においてさまざまな修正を施して本発明を利用することができるように、本発明の原理およびその実際的な応用について説明した。

【0194】したがって、本発明のいくつかの特徴についてしか具体的には説明しなかったが、明らかなように、本発明の技術思想および技術的範囲から離れることなく、さまざまな変形を行うことが可能である。さらに、頭字語は、単に、本明細書の可読性を高めるために用いられている。注意すべき点であるが、このような頭字語は、用いられている用語の一般性を狭めることを意

図しておらず、特許請求の範囲を限定すると解釈されてはならない。

【図面の簡単な説明】

【図1】マルチレイヤシミュレーション環境を有する従来のシミュレーションシステムを示す図である。

【図2】本発明の特徴による、電子回路と、その回路をターゲットとするソフトウェアとのコバリデーション方法の基本プロセスフローを示す図である。

【図3】本発明の特徴による、ハードウェアの動作記述をソフトウェアモデルに翻訳する方法の詳細なプロセスフローを示す図である。

【図4】本発明の特徴による、ハードウェアの動作記述をソフトウェアモデルに翻訳する方法の詳細なプロセスフローを示す図である。

【図5】命令セットシミュレータへのコマンドラインインタフェースの例示的なコマンドのセットを示す図である。

【図6】インテル1960プロセッサによる代表的な命令フローを示す図である。

【図7】本発明の特徴による、命令がターゲットマイクロプロセッサを通る際の命令のサイクリングを決定するプロセスフローを示す図である。

【図8】本発明の特徴による、命令がターゲットマイクロプロセッサを通る際の命令のサイクリングを決定するプロセスフローを示す図である。

【図9】本発明の特徴による、命令がターゲットマイクロプロセッサを通る際の命令のサイクリングを決定するプロセスフローを示す図である。

【図10】本発明の特徴による、軽量コンピュータオペレーティングシステム環境でコバリデーションシミュレーションが実行される場合の、エミュレートされるターゲットプロセッサのプロセスフローを示す図である。

【図11】本発明の特徴による、軽量コンピュータオペレーティングシステム環境でコバリデーションシミュレーションが実行される場合の、エミュレートされるターゲットプロセッサのプロセスフローを示す図である。

【図12】本発明の特徴による、軽量コンピュータオペレーティングシステム環境でコバリデーションシミュレーションが実行される場合の、エミュレートされるターゲットプロセッサのプロセスフローを示す図である。

【図13】本発明の特徴による、軽量コンピュータオペレーティングシステム環境でコバリデーションシミュレーションが実行される場合の、エミュレートされるターゲットプロセッサのプロセスフローを示す図である。

【図14】本発明の特徴による、軽量コンピュータシステム環境におけるコバリデーションの例示的なソフトウェアプロセスを示す図である。

【図15】本発明の特徴による、軽量コンピュータオペレーティングシステム環境における信号フローバスを示す図である。

【図 16】本発明の特徴による、ターゲットマイクロプロセッサおよび電気回路のハードウェアおよびソフトウェアコバリデーションのための例示的なコンピュータシステムを示す図である。

【図 17】本発明を用いて協調検証されるターゲットマイクロプロセッサ、グルーロジック、I/Oハードウェアおよびメモリからなる例示的なシステムを示す図である。

【図 18】図 17 に示した例示的なシステムのバスリードタイミングを示す図である。

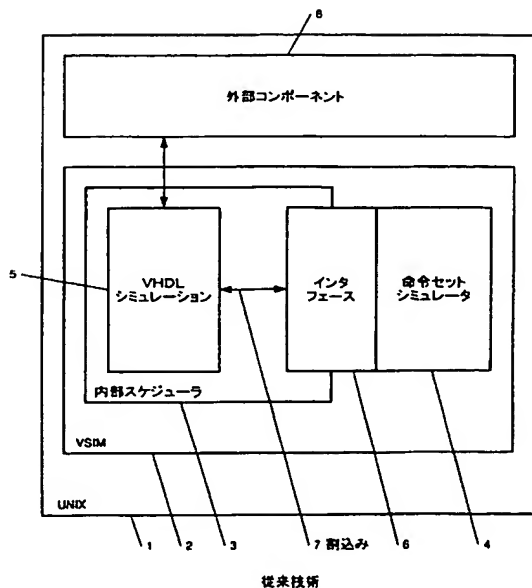
【図 19】図 17 に示した例示的なシステムのバスライトタイミングを示す図である。

【図 20】図 17 に示した例示的なシステムに対して、本発明によって生成される信号タイミングを示す図である。

【符号の説明】

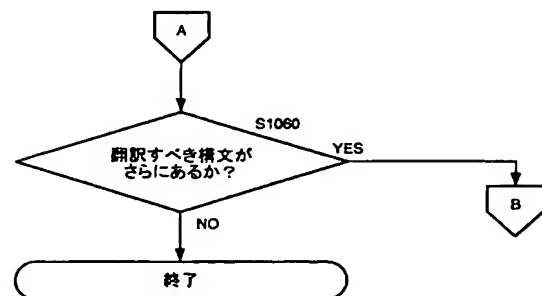
- 1 UNIXシステム
- 2 VSIM環境
- 3 内部スケジューラ
- 4 命令セットシミュレータ
- 5 VHDLシミュレーション
- 6 インタフェース
- 7 割込み
- 8 外部コンポーネント
- 10 ファブリックプロセス
- 11 中央制御プロセス
- 12 信号メンテナサブシステム

【図 1】

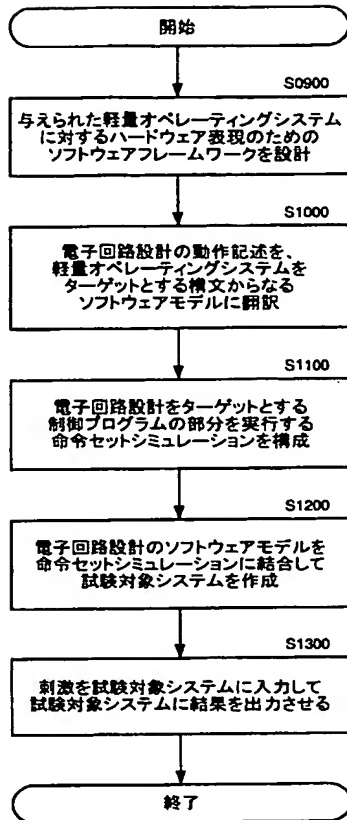


- 13 クロックジェネレータサブシステム
- 14 キューメンテナサブシステム
- 15 ディスプレージェネレータ
- 16 スケジューラサブシステム
- 21 試験対象回路
- 22 バスアービトレーションプロセス
- 23 グローバルループコントローラ
- 24 試験刺激プロセス
- 30 軽量オペレーティングシステム
- 31 ファブリックプロセス
- 32 命令セットシミュレーションプロセス
- 33 VHDLプロセス
- 34 VHDLプロセス
- 35 I960プロセッサ
- 36 グルーロジック
- 37 UART
- 38 メモリサブシステム
- 40 プロセッサ
- 41 ビデオディスプレイ端末
- 42 メモリ
- 43 I/Oデバイス
- 44 データリンク
- 45 サーバ
- 46 プログラムライブラリ
- 50 開発ボード
- 51 通信媒体

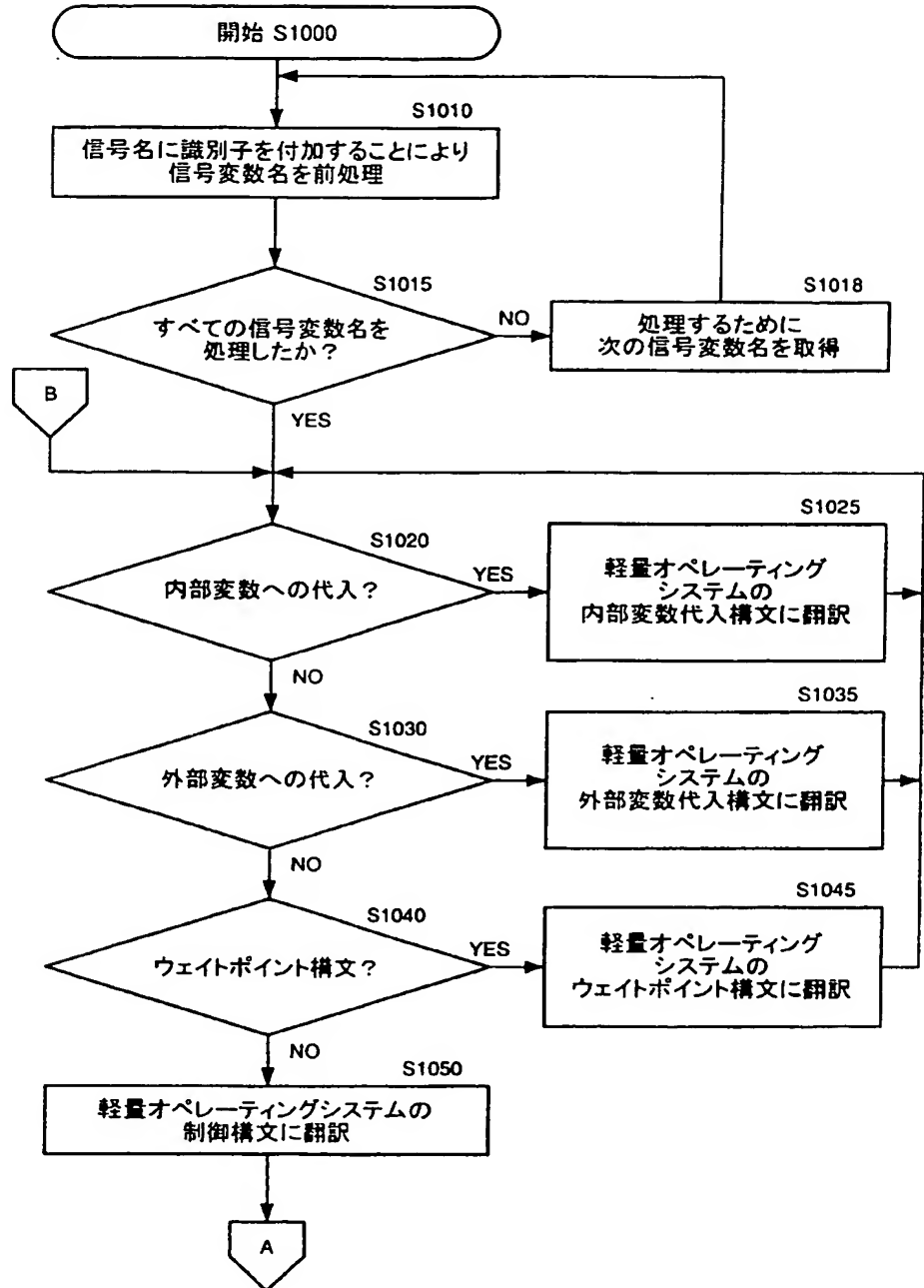
【図 4】



【図 2】



【図 3】

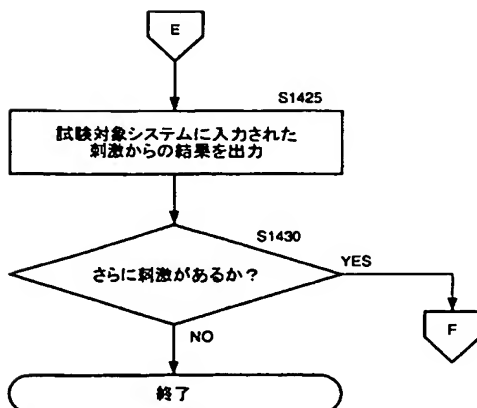


【図 5】

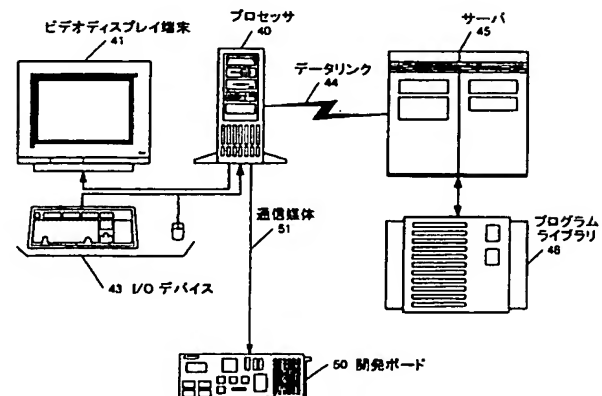
コマンド	アクション
All	すべてのレジスタ／状態変数を表示する
Stats	統計処理を実行する
ICounts	命令ごとのカウンタ
Mem <a> <w>	アドレス<a>から<v>まで幅<w>でメモリを表示する
Set <r> <v>	レジスタ<r>を値<v>にセットする
Show <r>	レジスタ<r>の値を印字する
Step	シングルステップ 1 命令
Go <n>	たかだか<n>個の命令だけステップする
Break <a>	アドレス<a>にブレークポイントをセットする
IStop [on off]	割り込みで停止する
Print [on off]	命令ごとの印字をセットする
Time [on off]	命令ごとのタイミングをセットする
Int <n>	割り込み<n>を発生する
XLoad <f> [B]	ターゲットファイル<f>をロードする (B はビッグエンディアン)
Map <f>	ターゲットシンボルファイル<f>をロードする
Local <f> <a> <s> [Clear]	ローカルメモリを宣言する
Hardware <n> <a> <s>	メモリマップド I/O 領域を宣言する
State <f> [Clear]	状態変数用ファイルを宣言する
Obey <f>	<f>に格納されたコマンドに従う
Quit	ISSを終了する
Help	このテキストを印字する

テーブル 1. ISS へのコマンドラインインタフェース

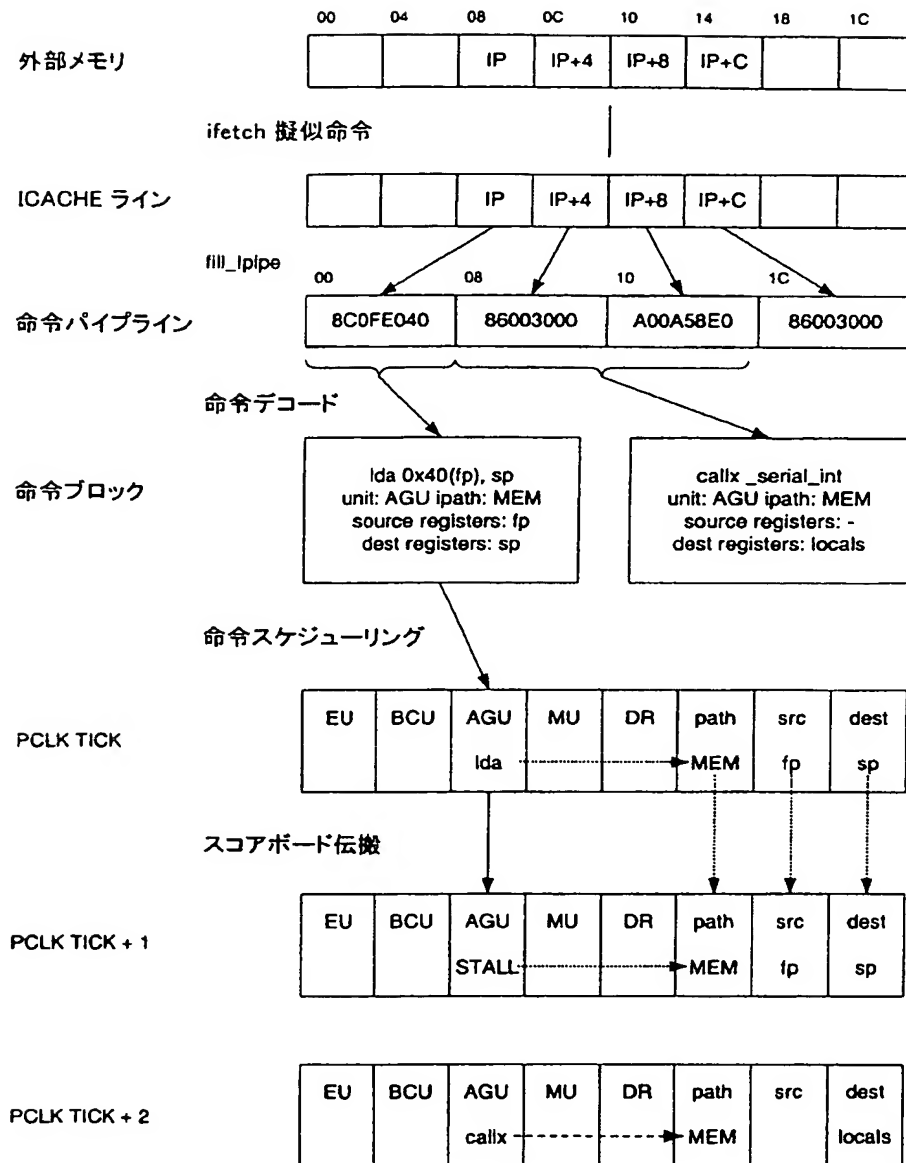
【図 13】



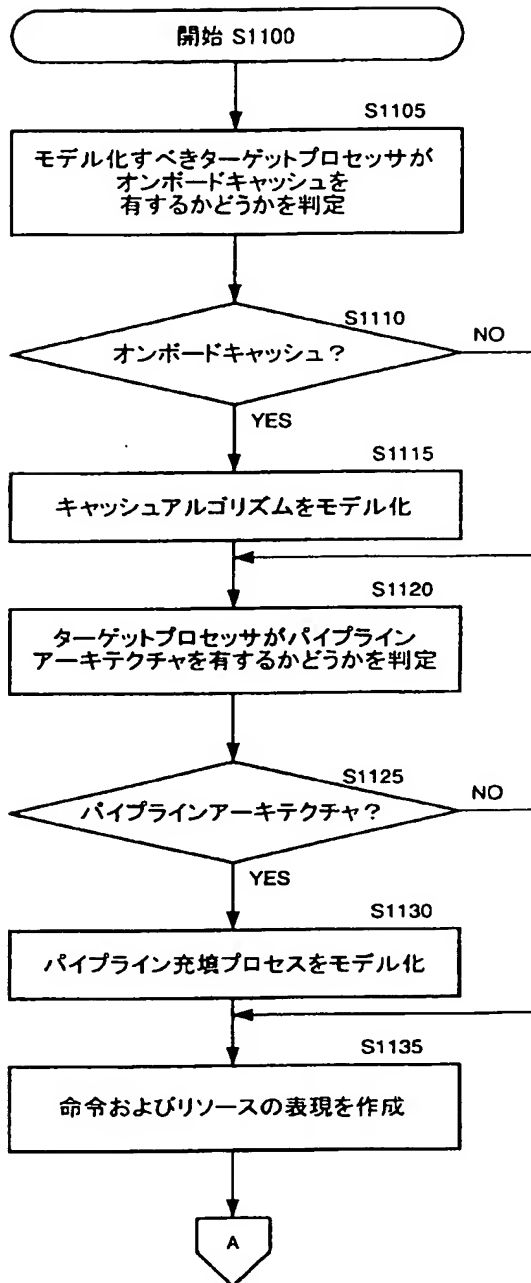
【図 16】



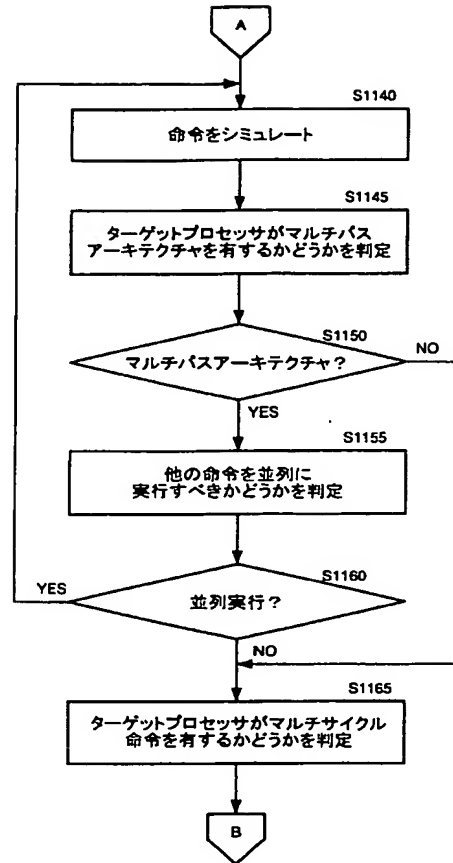
【図6】



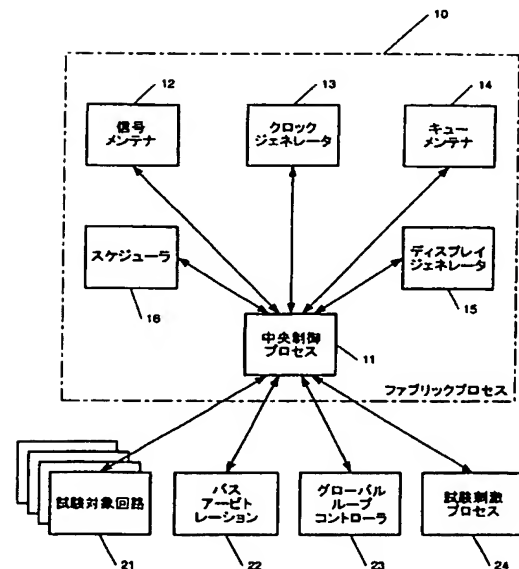
【図 7】



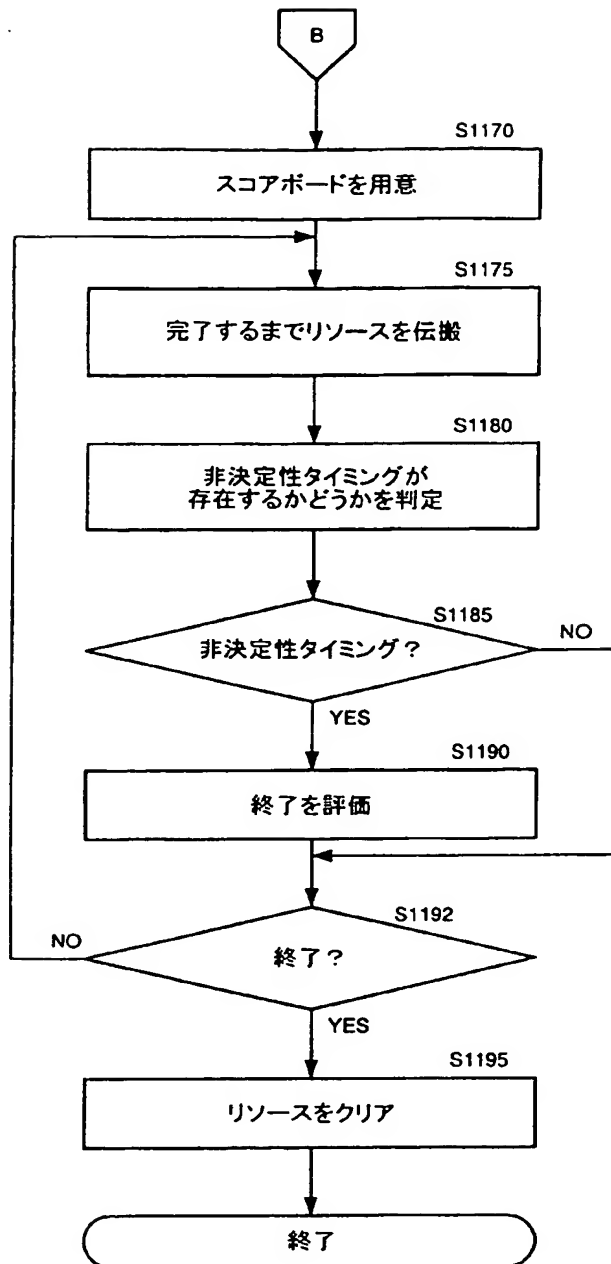
【図 8】



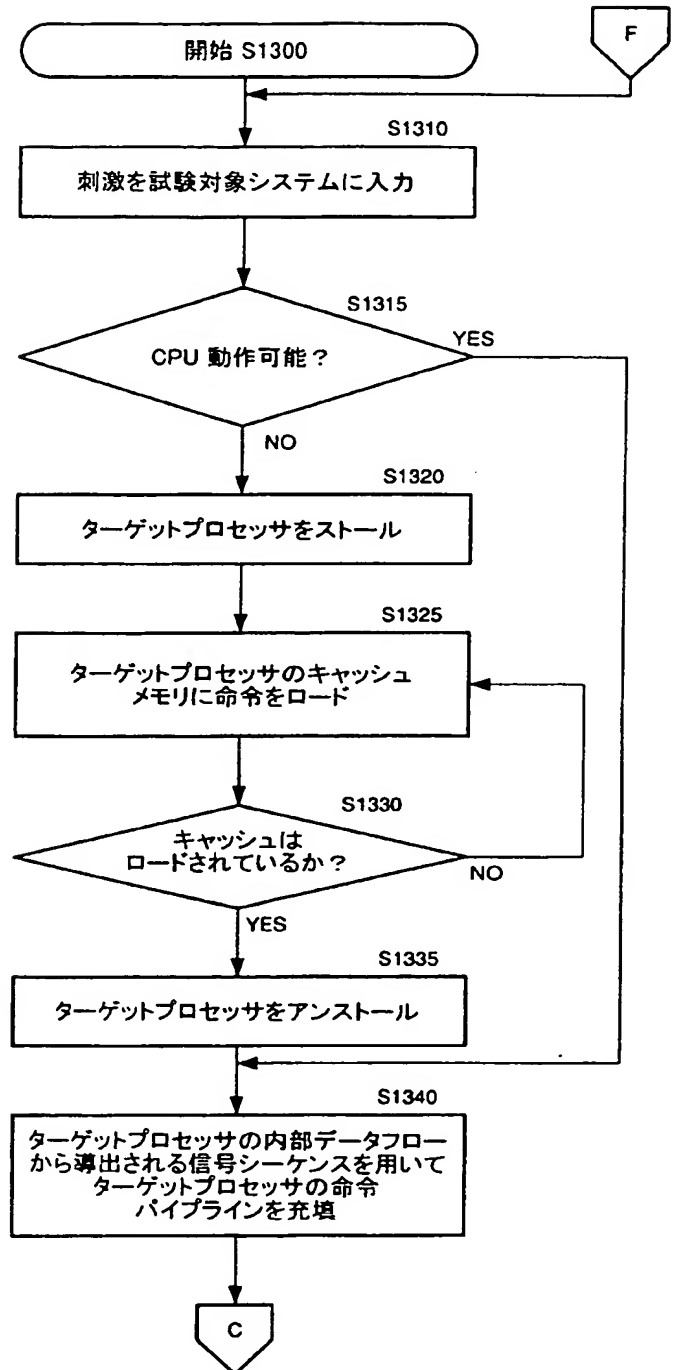
【図 14】



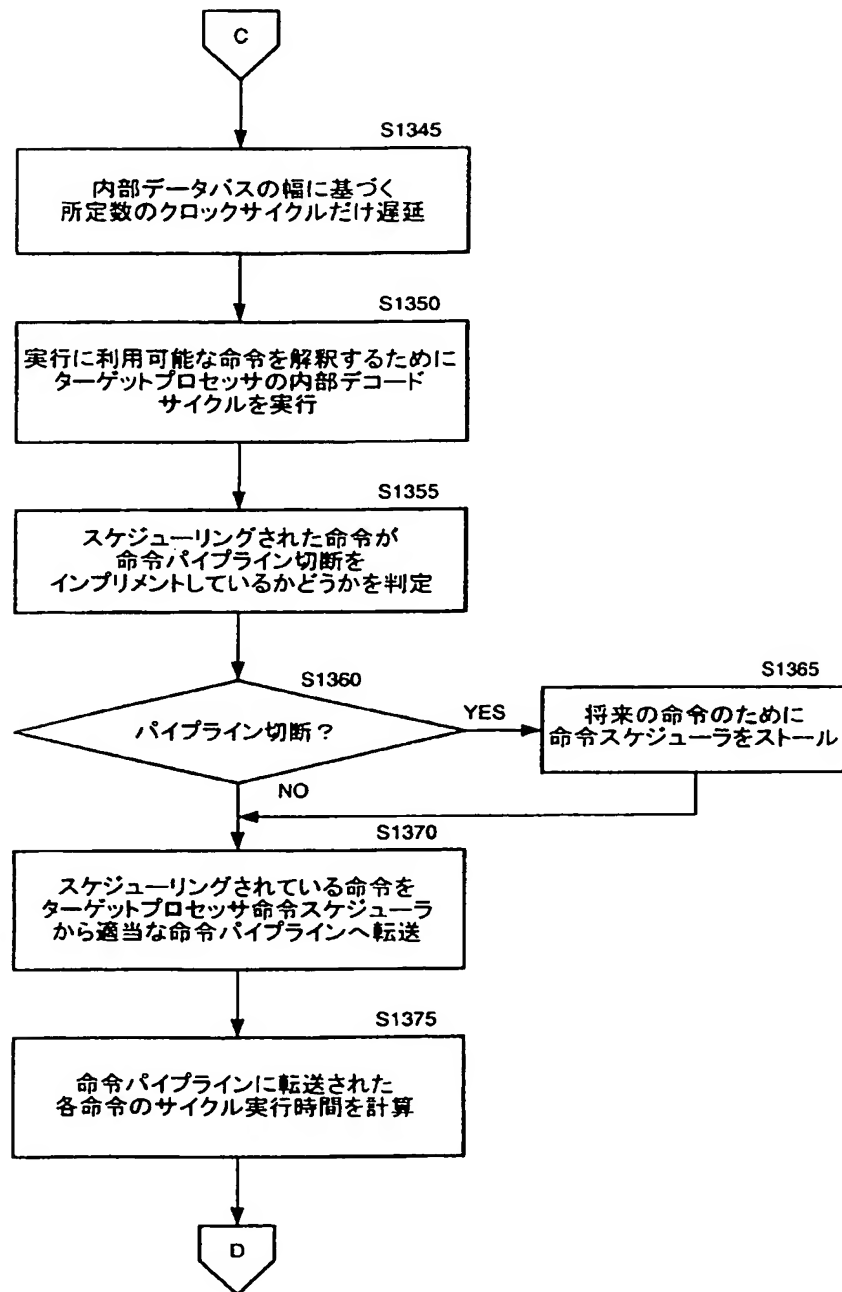
【図 9】



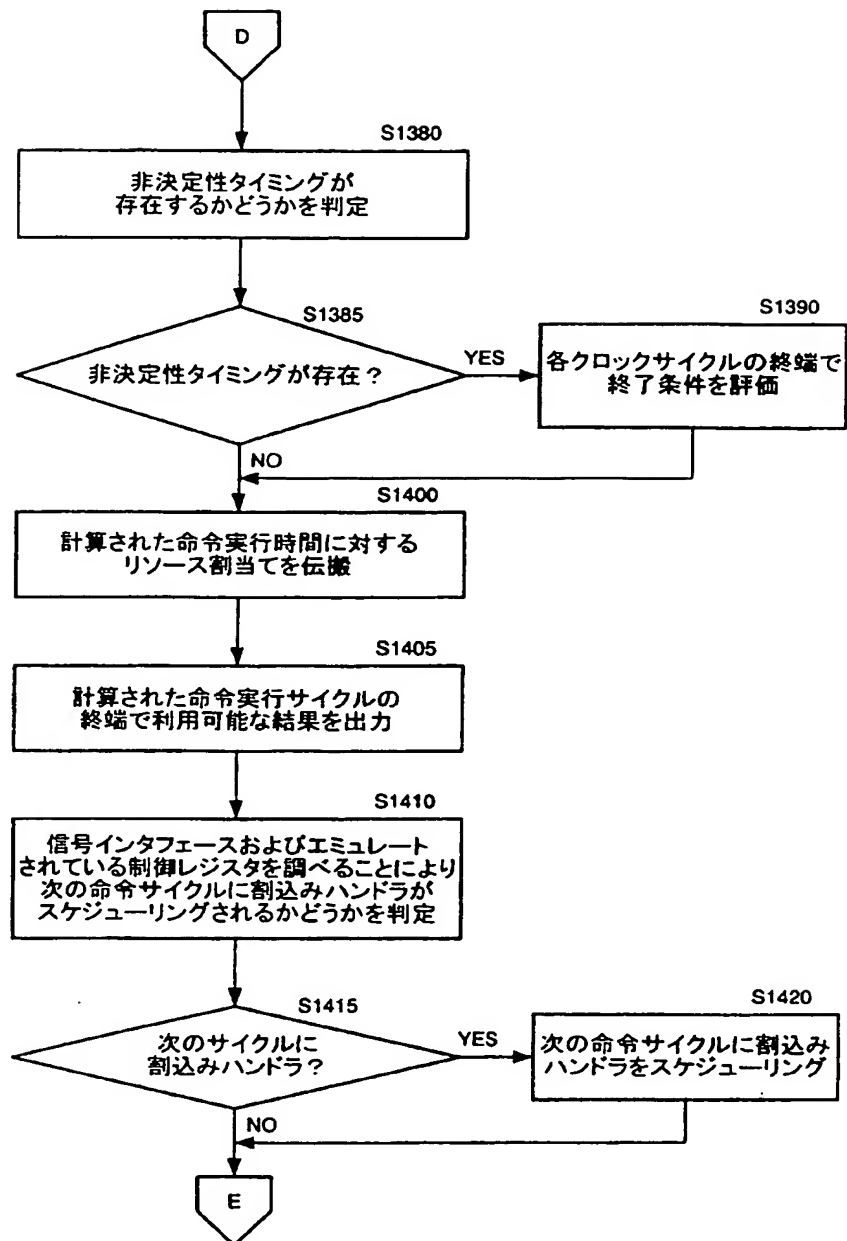
【図 10】



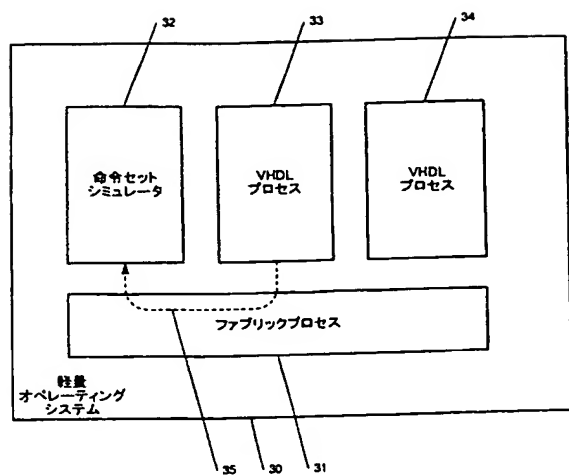
【図 11】



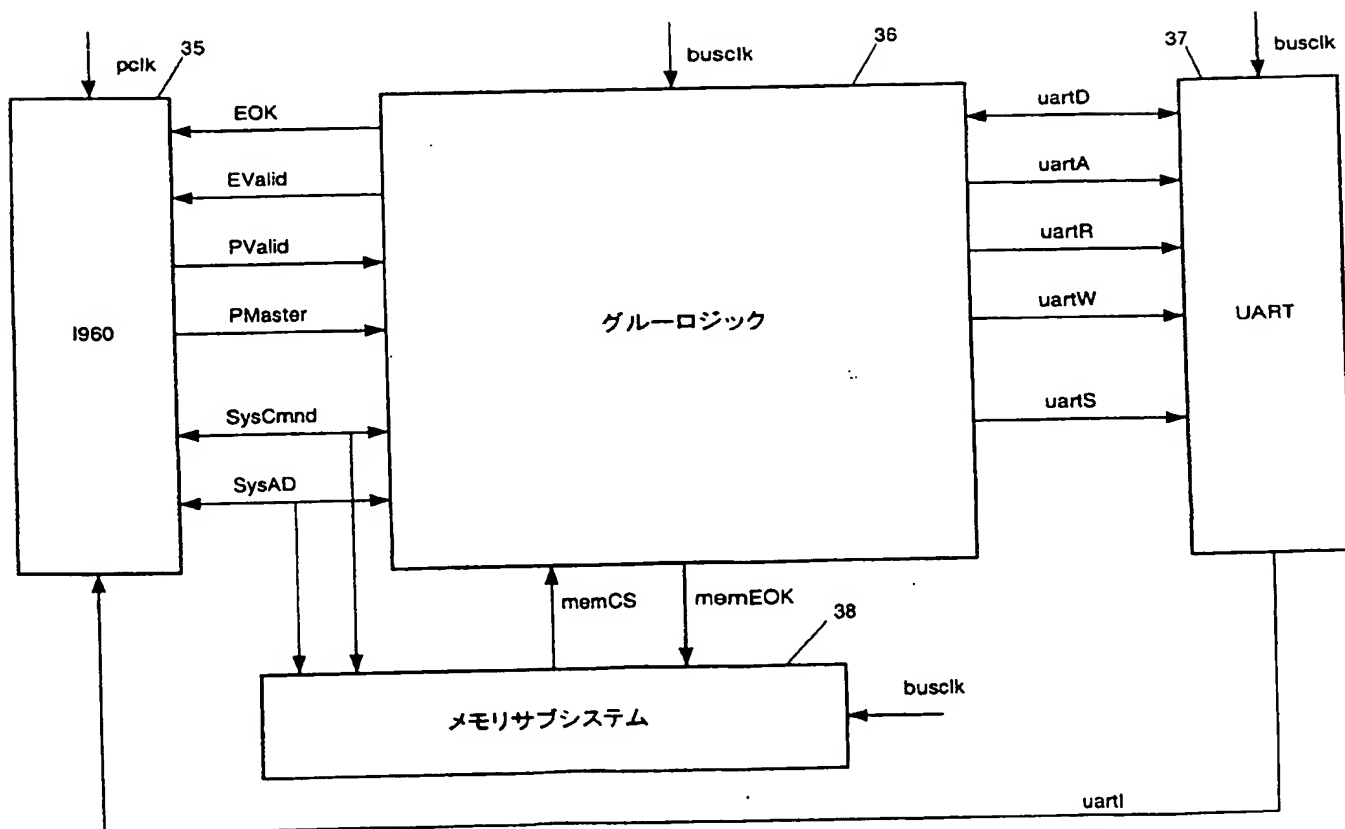
【図12】



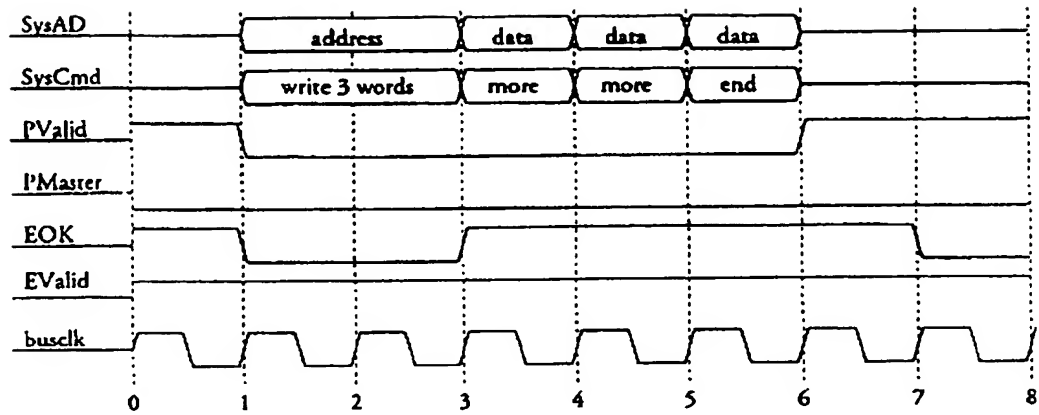
【図15】



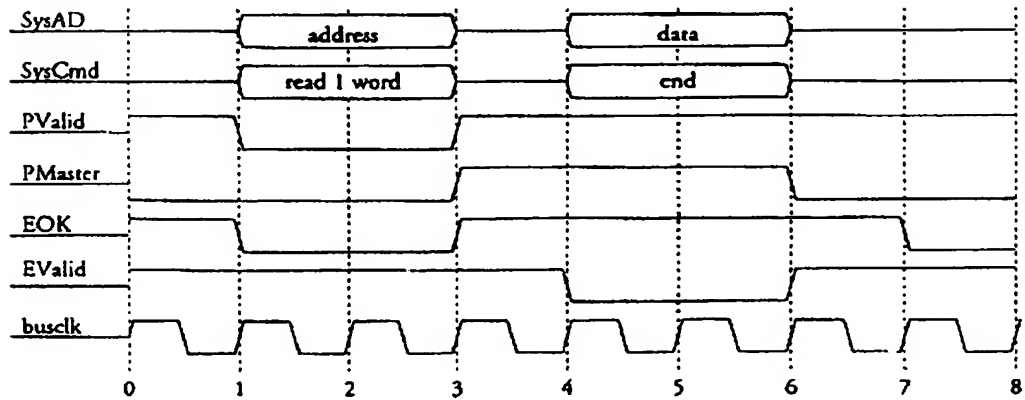
【図17】



【図18】



【図19】



【図20】

